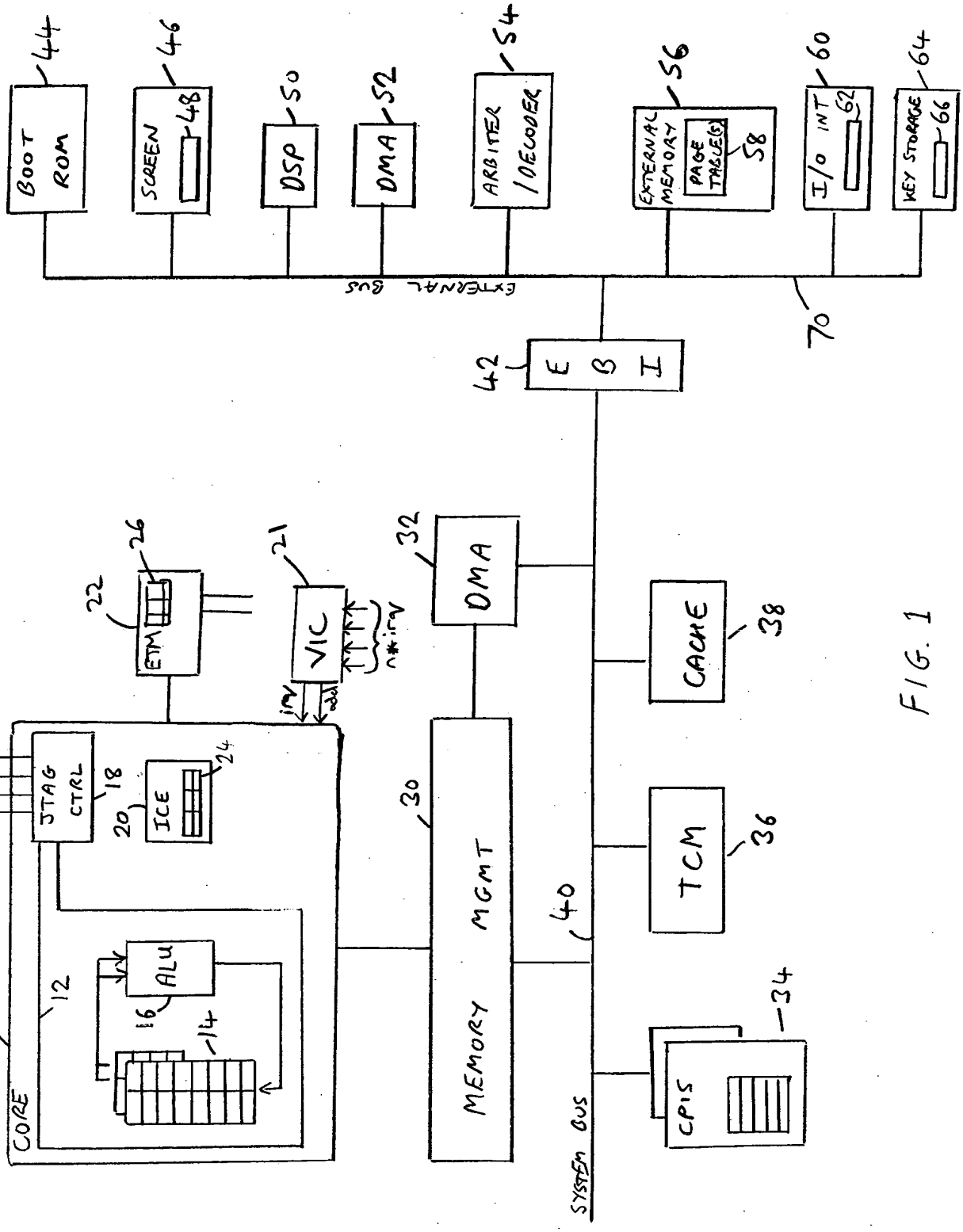

BOOT ROM
44
SCREEN
46
48
DSP
50
DMA
52
ARBITER /DECODER
54
EXTERNAL MEMORY
56
PAGE TABLE(S)
58
I/O INT
60
62
KEY STORAGE
64
66
EXTERNAL BUS
70
42
E B I
32
DMA
22
ETM
26
21
VIC
IRQ
FIQ
JTAG CTRL
18
20
ICE
24
ALU
16
14
12
10
CORE
30
MEMORY MGMT
40
SYSTEM BUS
CACHE
38
TCM
36
CPIS
34

FIG. 1

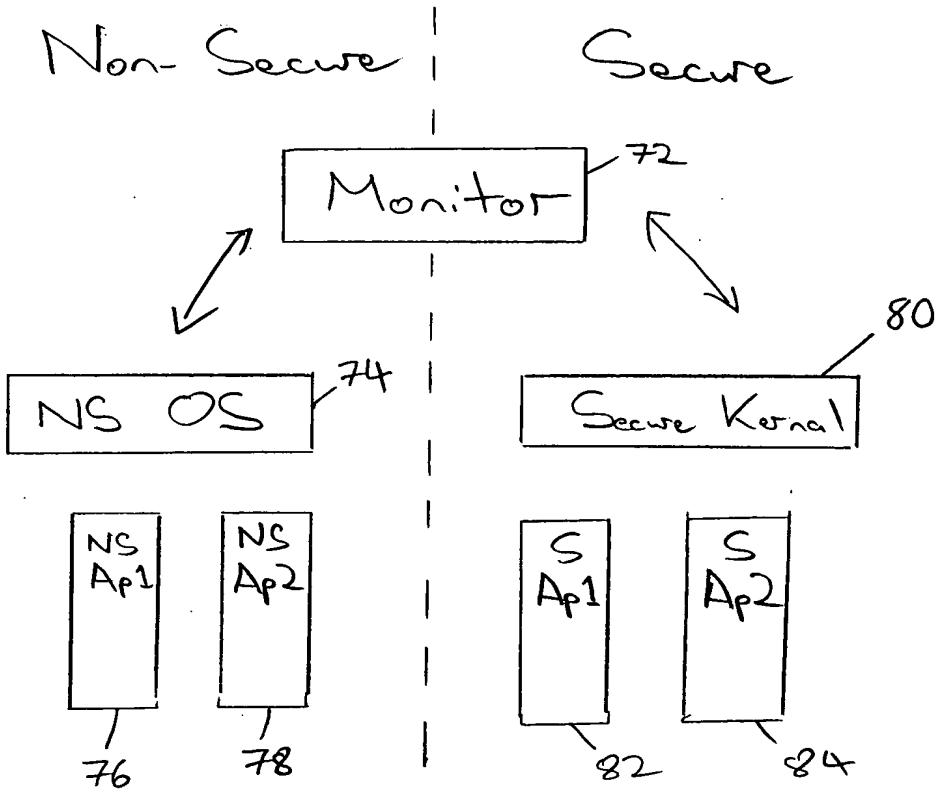

Non-Secure
Secure
Monitor
72
NS OS
74
Secure Kernal
80
NS Ap1
NS Ap2
S Ap1
S Ap2
76
78
82
84

Fig. 2

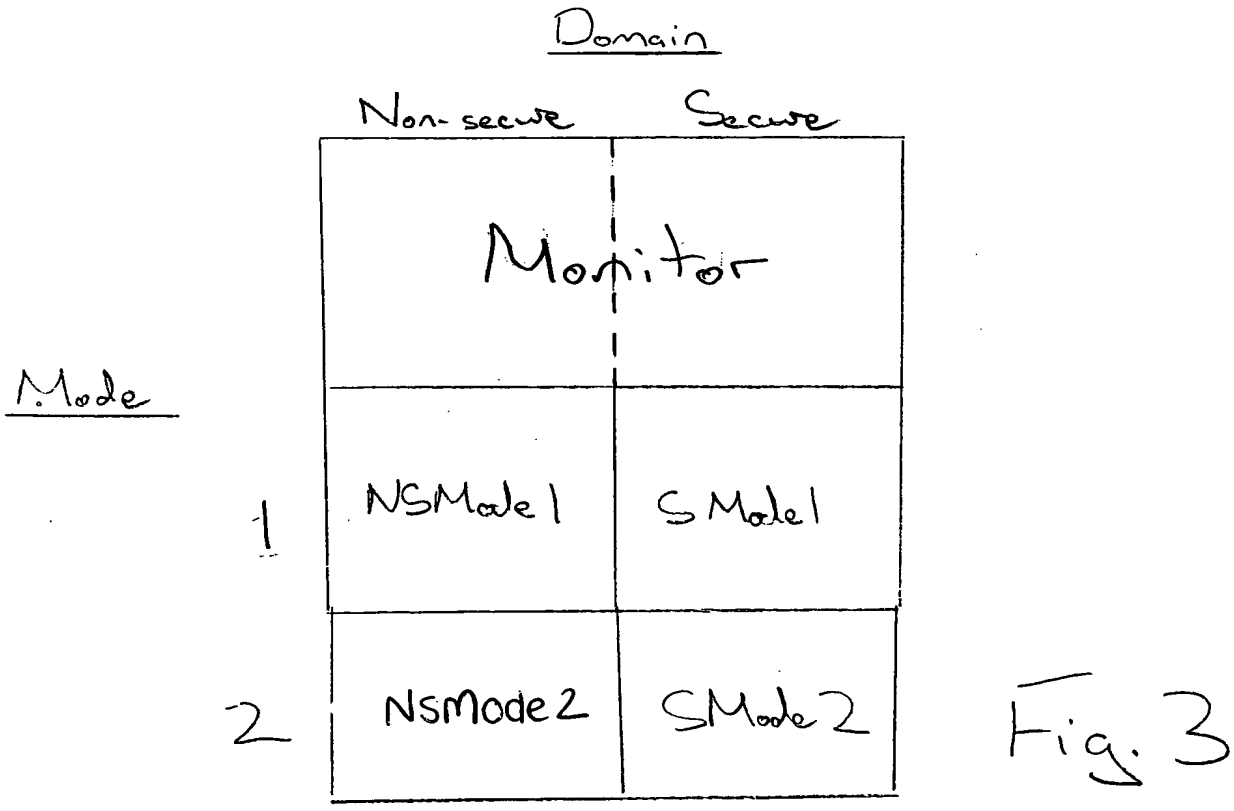

Domain
Non-secure
Secure
Monitor
Mode
1
NSMode1
SMode1
2
NSmode2
SMode2

Fig. 3

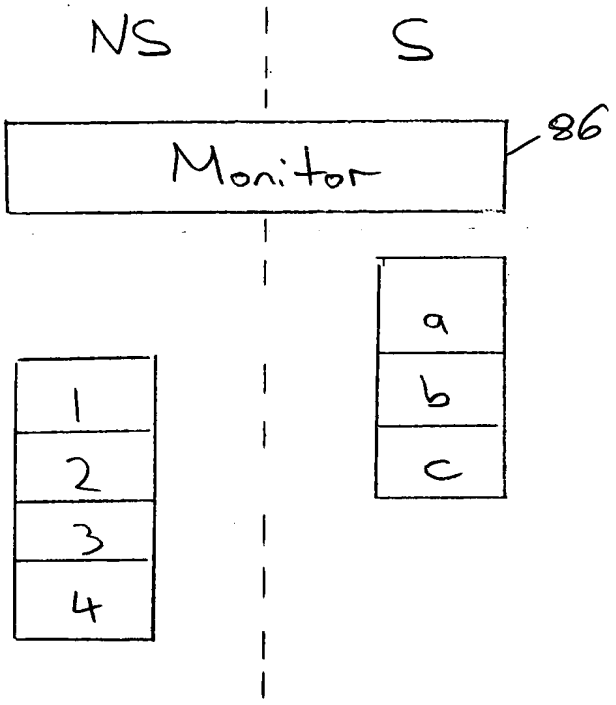


Fig. 4

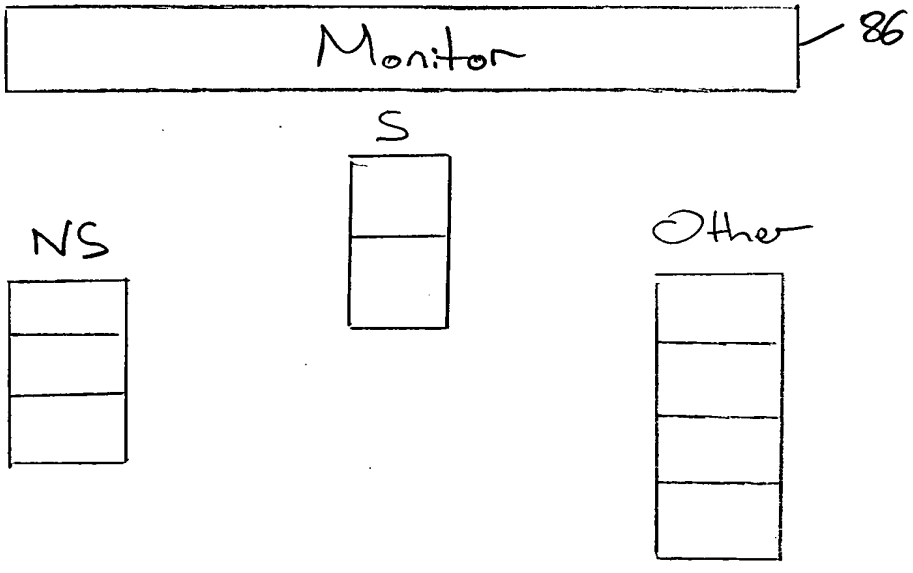


Fig. 5

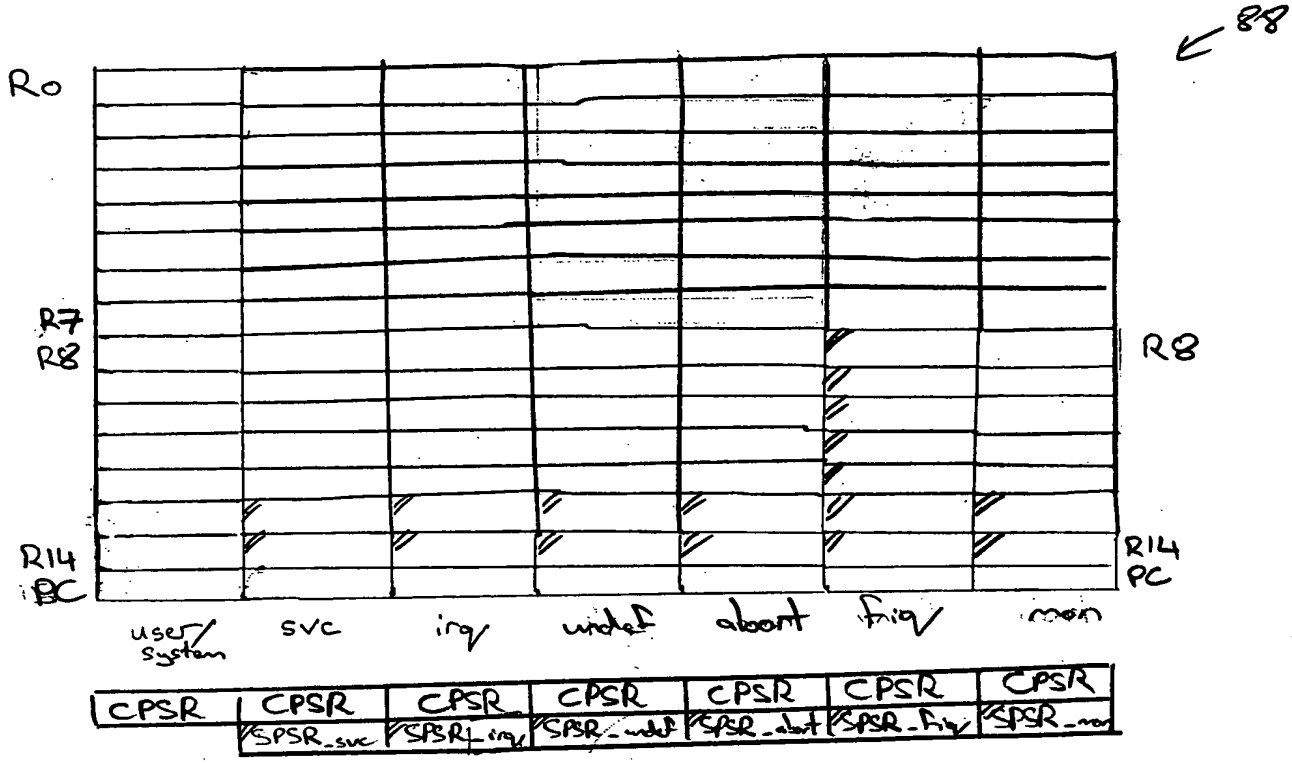

88
R0
R7
R8
R8
R14
PC
R14
PC
user/system
svc
irq
undef
abort
fiq
mon
CPSR
CPSR
SPSR_svc
CPSR
SPSR_irq
CPSR
SPSR_und
CPSR
SPSR_abt
CPSR
SPSR_fiq
CPSR
SPSR_mon
// = private to mode

Fig. 6

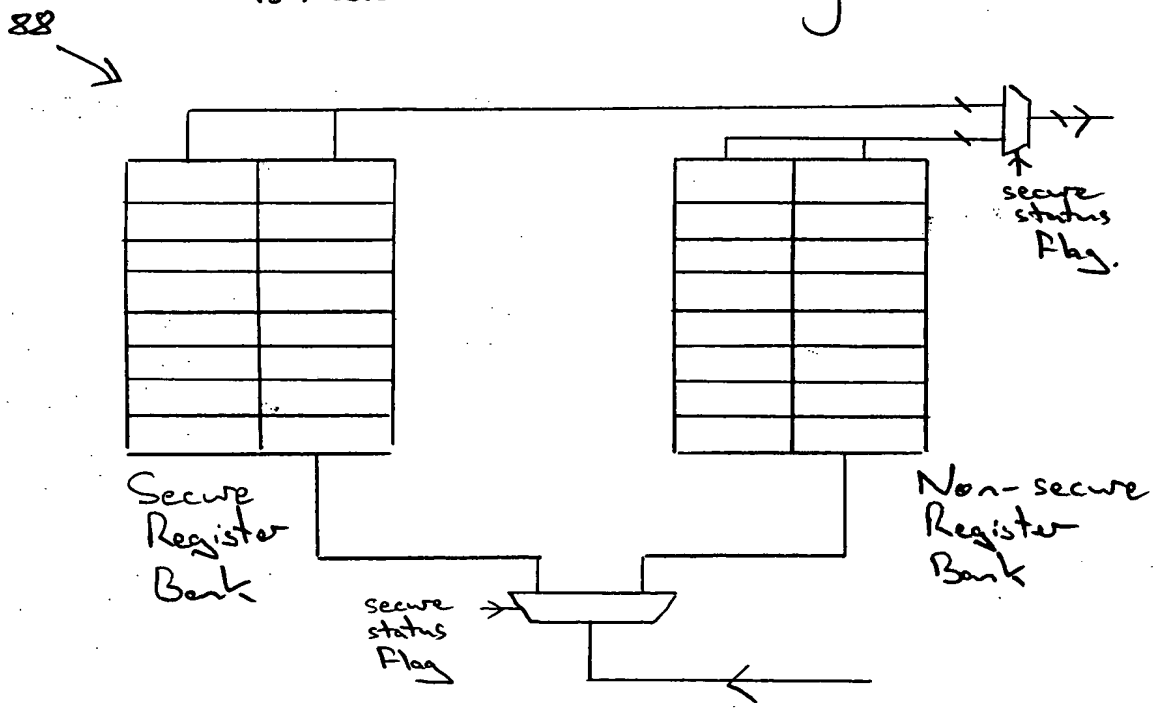

88
secure status Flag.
Secure Register Bank
Non-secure Register Bank
secure status Flag

Fig. 7

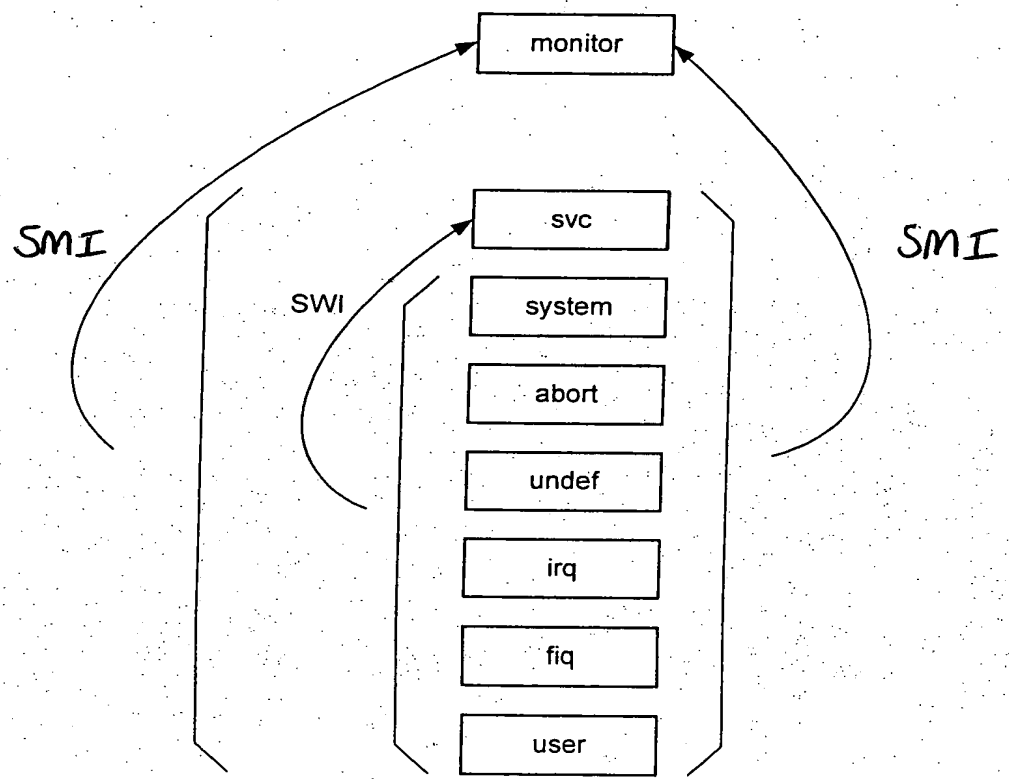


Fig. 8

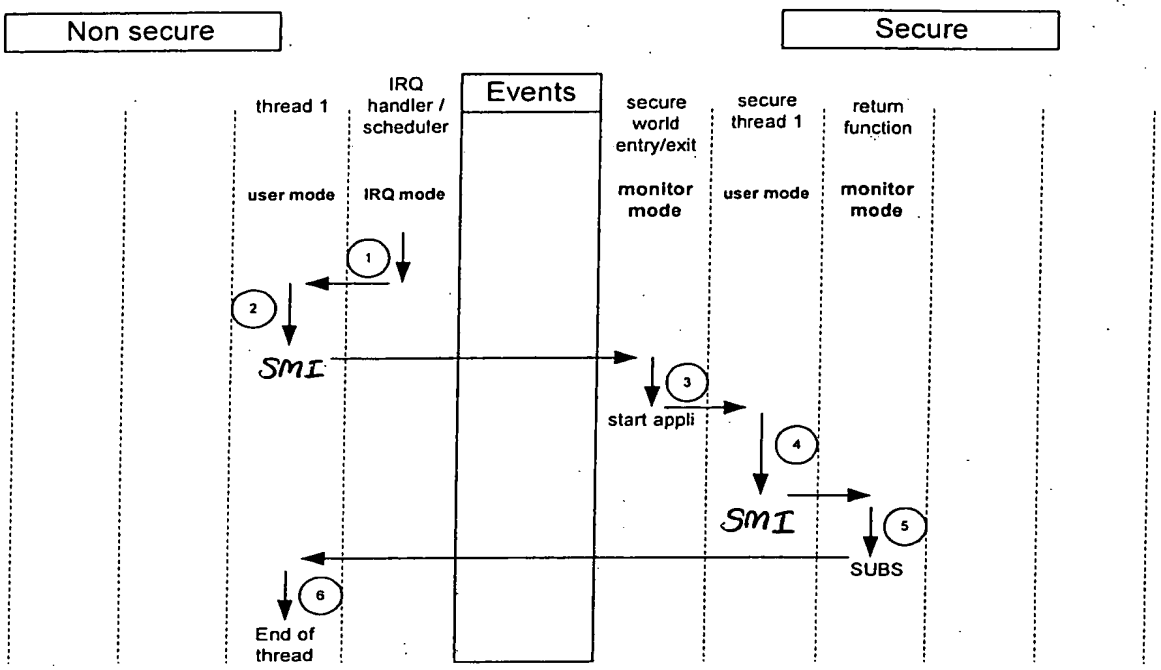

Non secure
Secure
thread 1
IRQ handler / scheduler
Events
secure world entry/exit
secure thread 1
return function
user mode
IRQ mode
monitor mode
user mode
monitor mode
1
2
SMI
3
start appli
4
SMI
5
SUBS
6
End of thread

Fig. 9

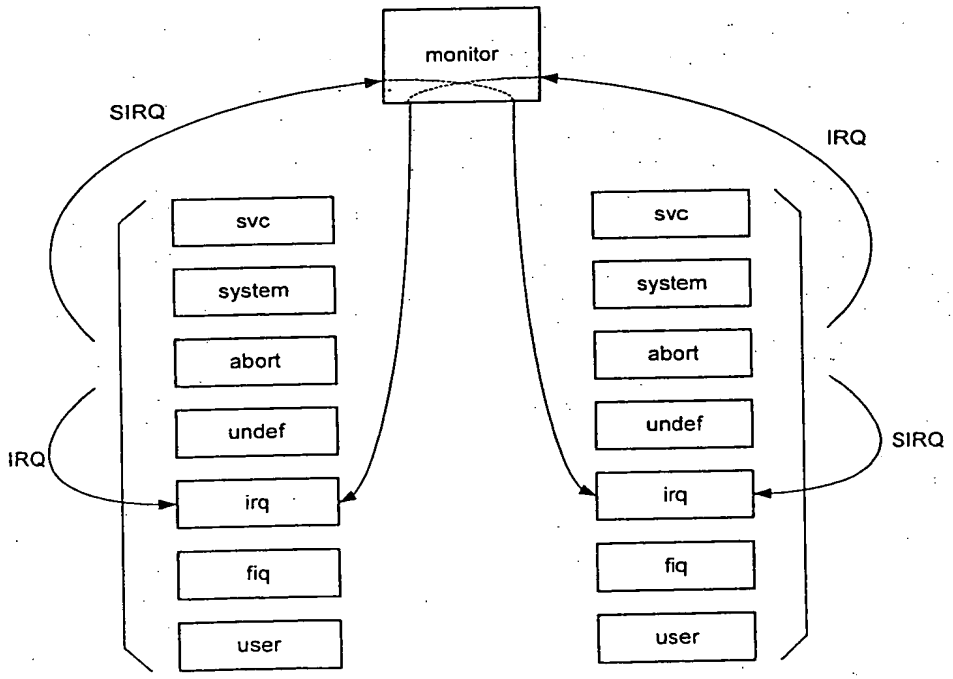

monitor
SIRQ
IRQ
svc
system
abort
undef
irq
fiq
user
svc
system
abort
undef
irq
fiq
user
IRQ
SIRQ

Fig. 10

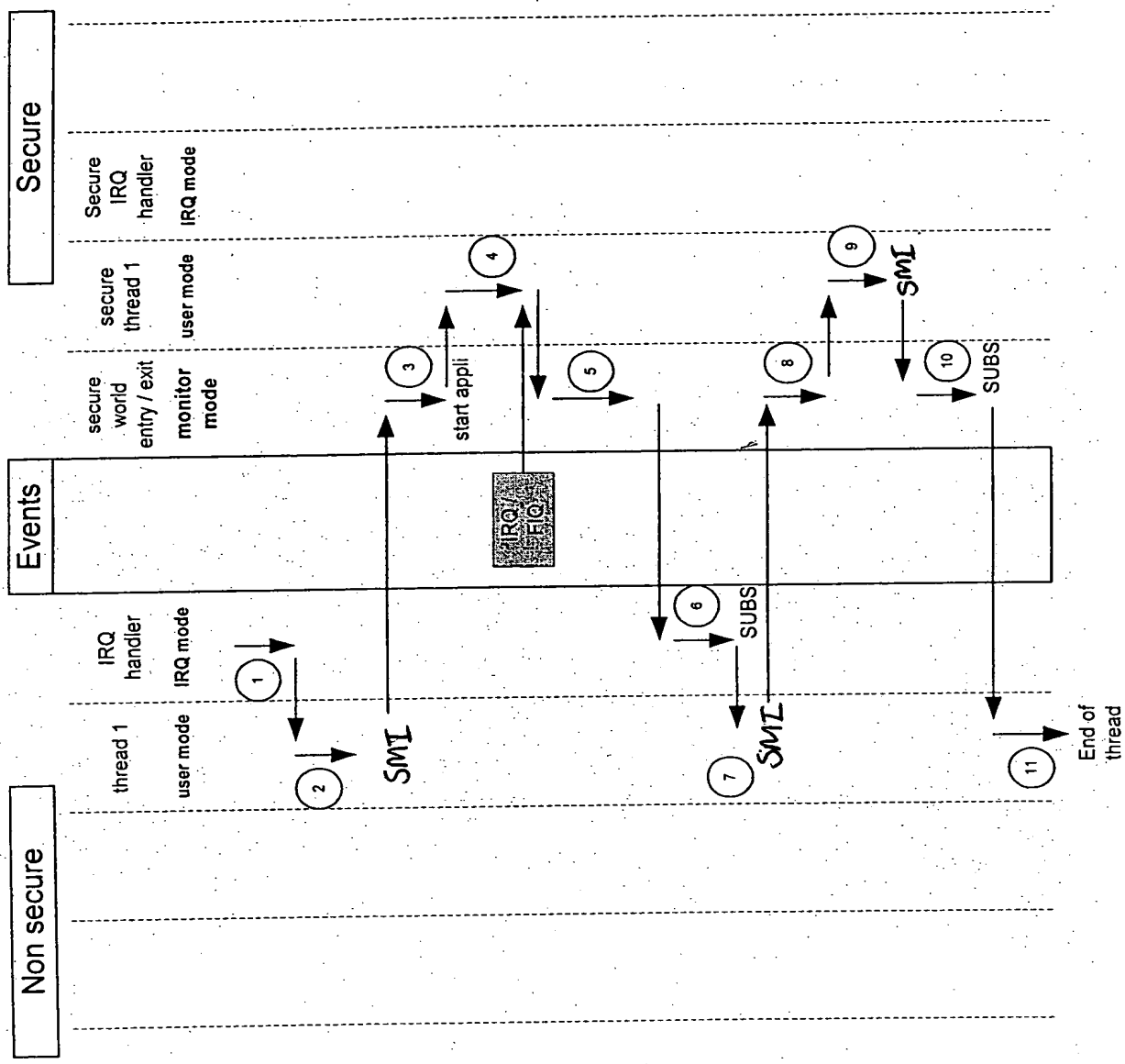

Non secure
Events
Secure
thread 1
user mode
IRQ handler
IRQ mode
secure world entry / exit
monitor mode
secure thread 1
user mode
Secure IRQ handler
IRQ mode
1
2
SMI
3
start appli
4
5
IRQ/ FIQ
6
SUBS
7
SMI
8
9
SMI
10
SUBS
11
End of thread

Fig 11A

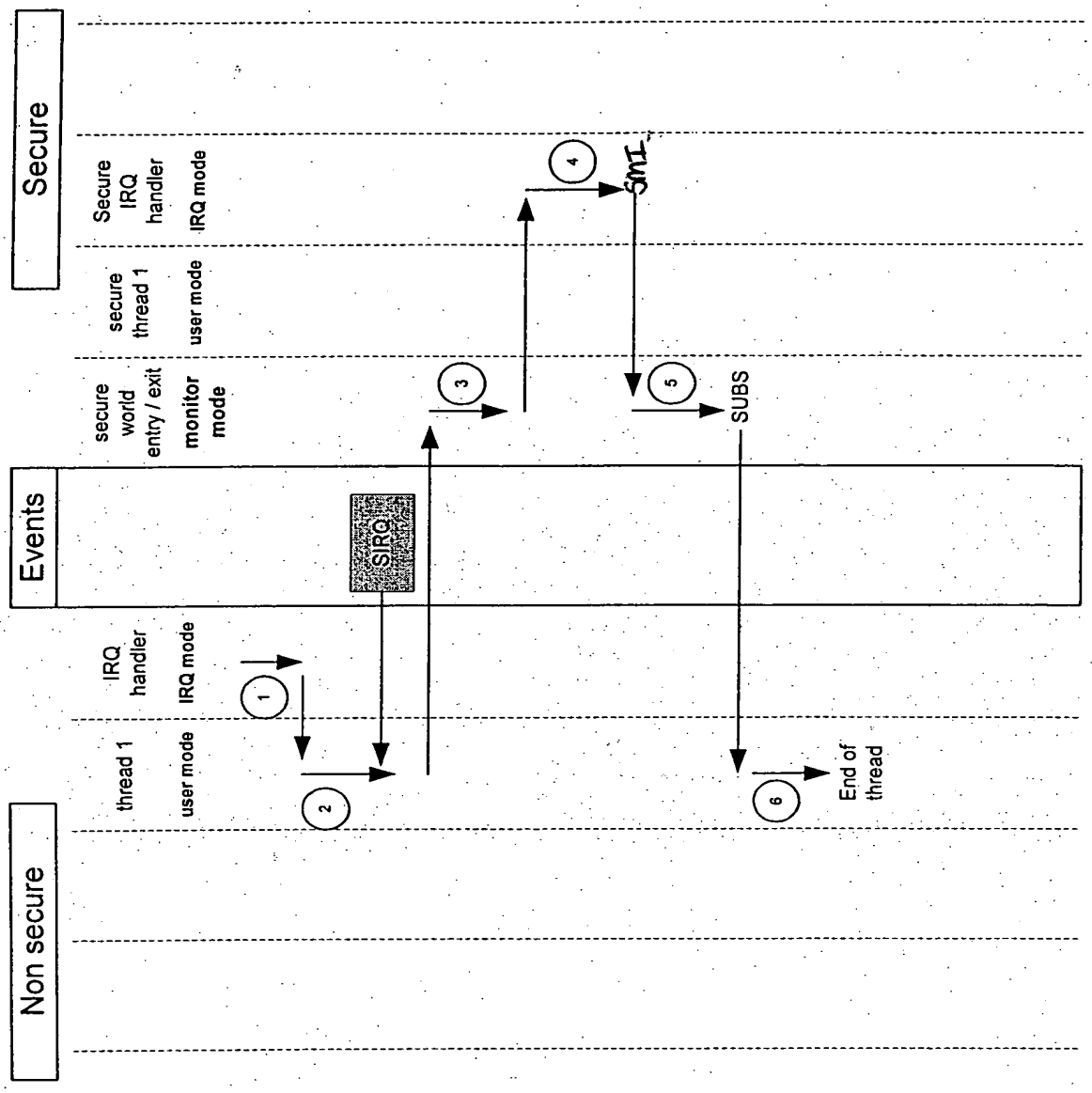

Secure
Non secure
Events
Secure IRQ handler
IRQ mode
secure thread 1
user mode
secure world entry / exit
monitor mode
IRQ handler
IRQ mode
thread 1
user mode
SIRQ
SMI
SUBS
End of thread
1
2
3
4
5
6

Fig 11B

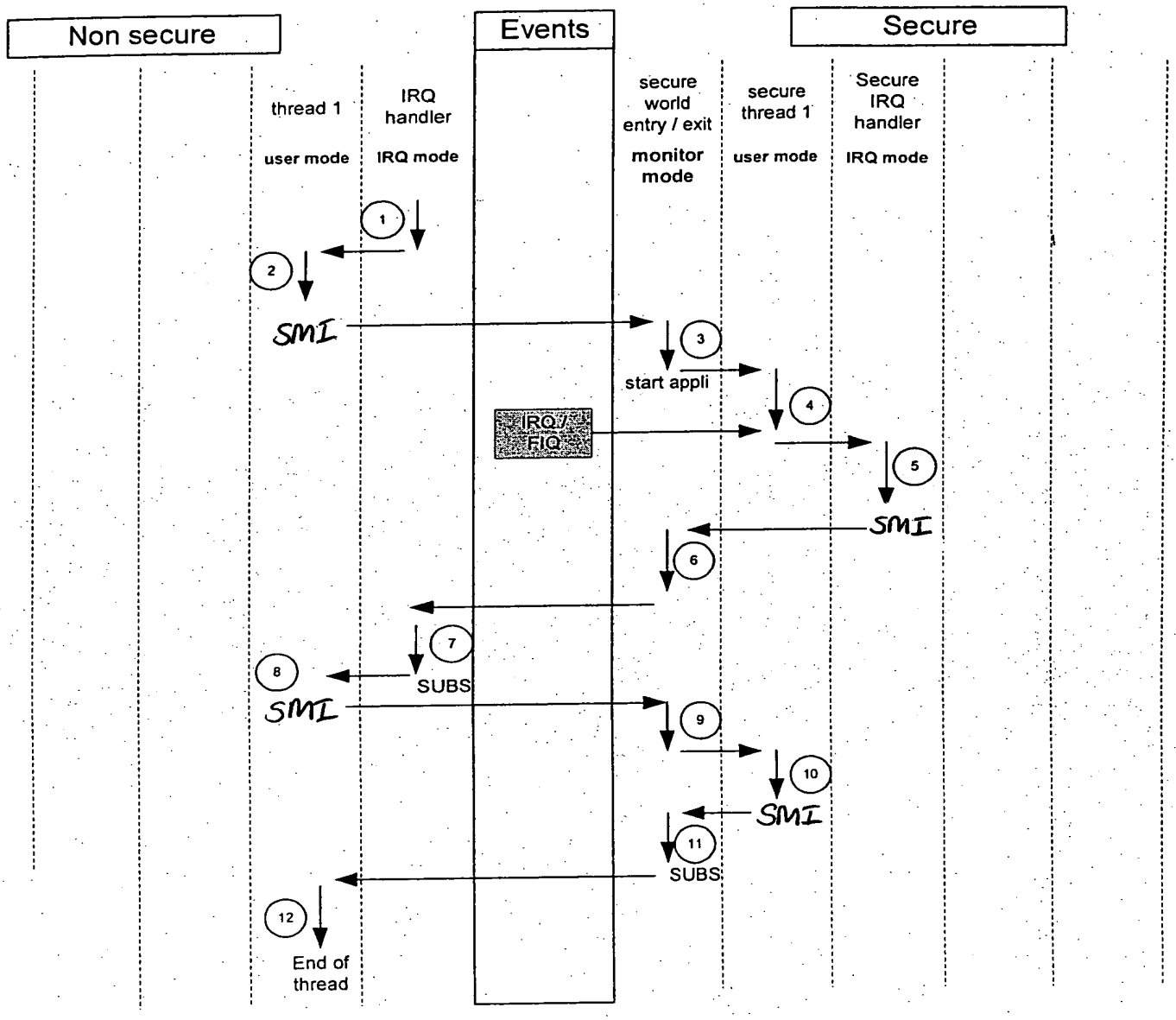

Non secure
Events
Secure
thread 1
user mode
IRQ handler
IRQ mode
secure world entry / exit
monitor mode
secure thread 1
user mode
Secure IRQ handler
IRQ mode
1
2
SMI
3
start appli
4
IRQ / FIQ
5
SMI
6
7
SUBS
8
SMI
9
10
SMI
11
SUBS
12
End of thread

Fig. 13A

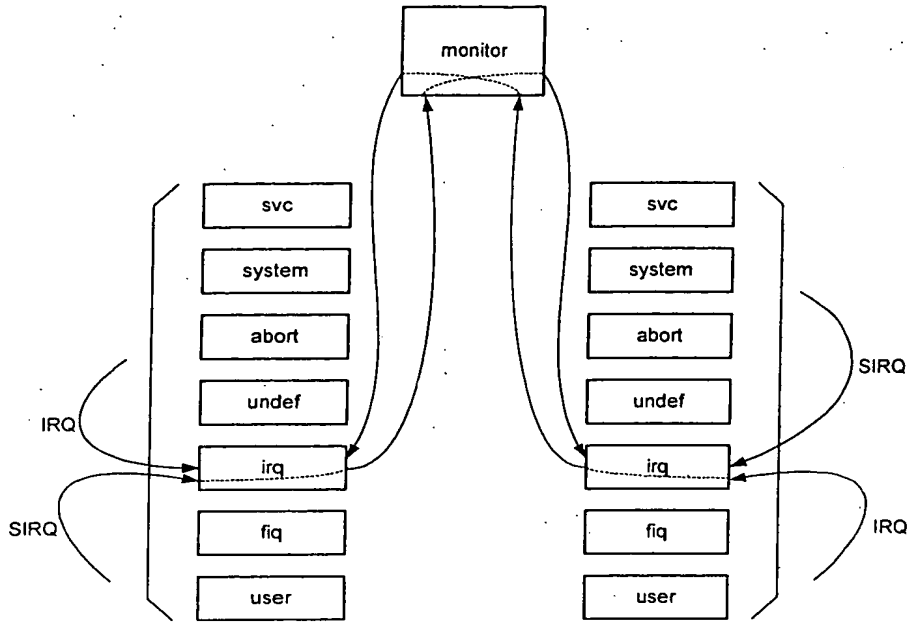


Fig. 12

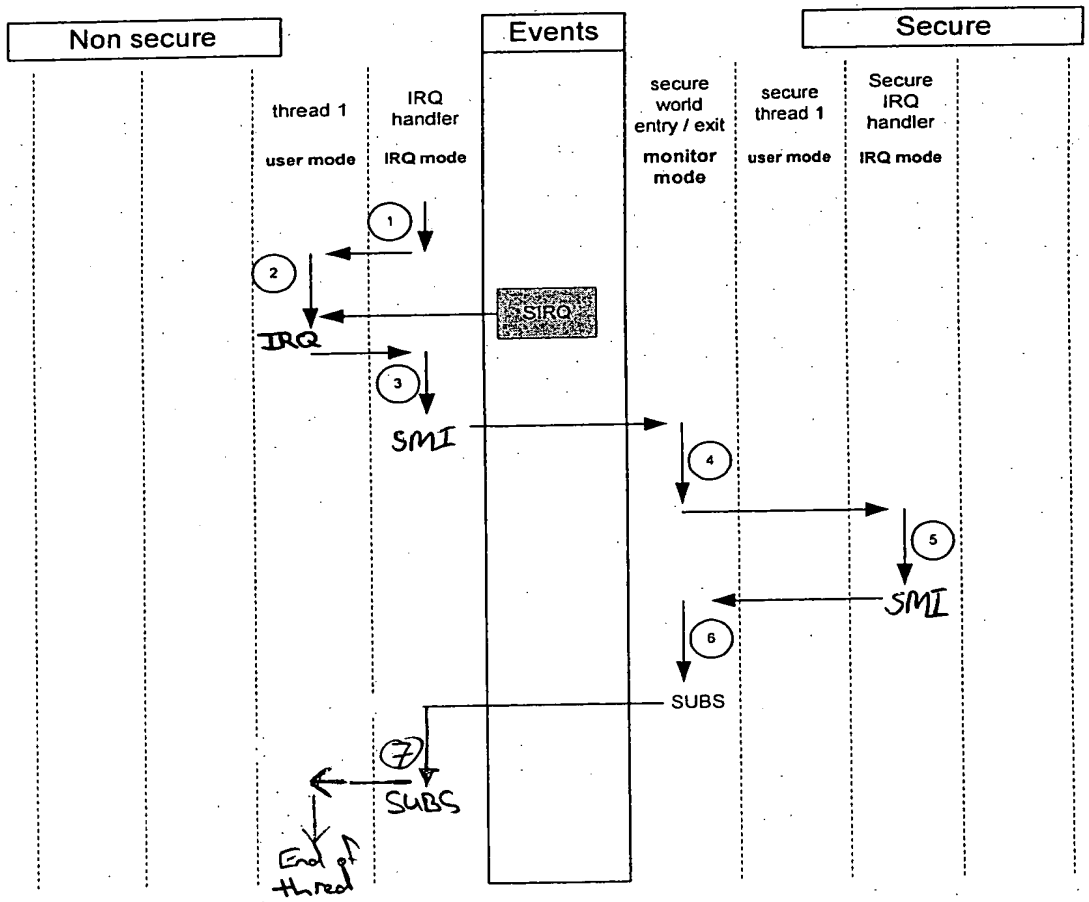


Fig. 13B

Exception	Vector offset	Corresponding mode
Reset	0x00	Supervisor mode
Undef	0x04	Monitor mode / Undef mode
SWI	0x08	Supervisor mode / Monitor mode
Prefetch abort	0x0C	Abort mode / Monitor mode
Data abort	0x10	Abort mode / Monitor mode
IRQ / SIRQ	0x18	IRQ mode / Monitor mode
FIQ	0x1C	FIQ mode / Monitor mode
SMI	0x20	Undef mode / Monitor mode

Fig. 14

Monitor

Reset	VM0
Undef	VM1
SWI	VM2
Prefetch abort	VM3
Data abort	VM4
IRQ/SIRQ	VM5
FIQ	VM6
SMI	VM7

Secure

Reset	VS0
Undef	VS1
SWI	VS2
Prefetch abort	VS3
Data abort	VS4
IRQ/SIRQ	VS5
FIQ	VS6
SMI	VS7

Non-Secure

Reset	VNS0
Undef	VNS1
SWI	VNS2
Prefetch abort	VNS3
Data abort	VNS4
IRQ/SIRQ	VNS5
FIQ	VNS6
SMI	VNS7

Fig. 15

CP15 Monitor Trap Mask Register

only NS world exceptions illustrated

SMI	SWI	Prefetch Abort	Data Abort	IRQ	SIRQ	FIQ
0	1	1	1	1	0	1

OR via hardware/external pin

1 = Mon(S)

0 = NS

Fig. 16

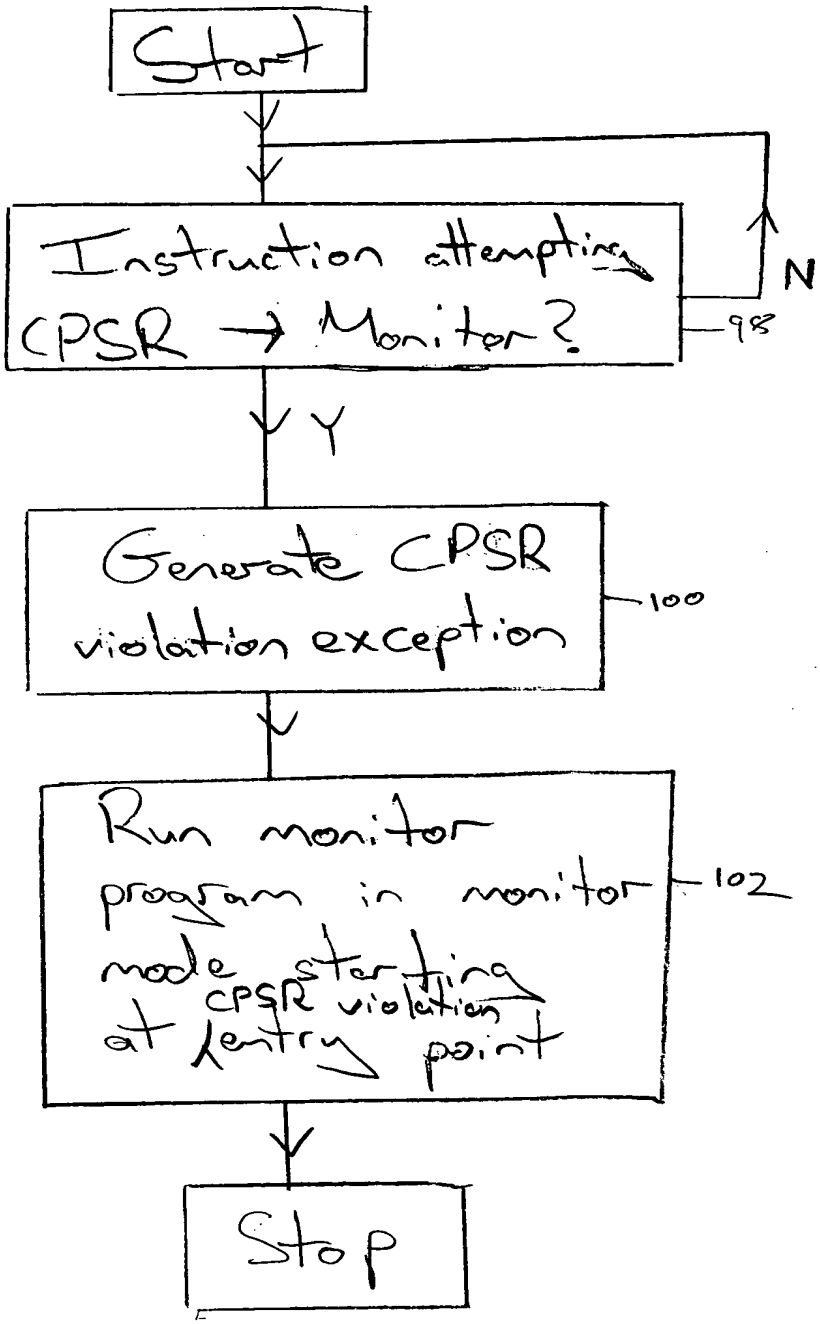

Start
Instruction attempting CPSR → Monitor?
N
98
Y
Generate CPSR violation exception
100
Run monitor program in monitor mode starting at CPSR violation entry point
102
Stop

Fig. 17

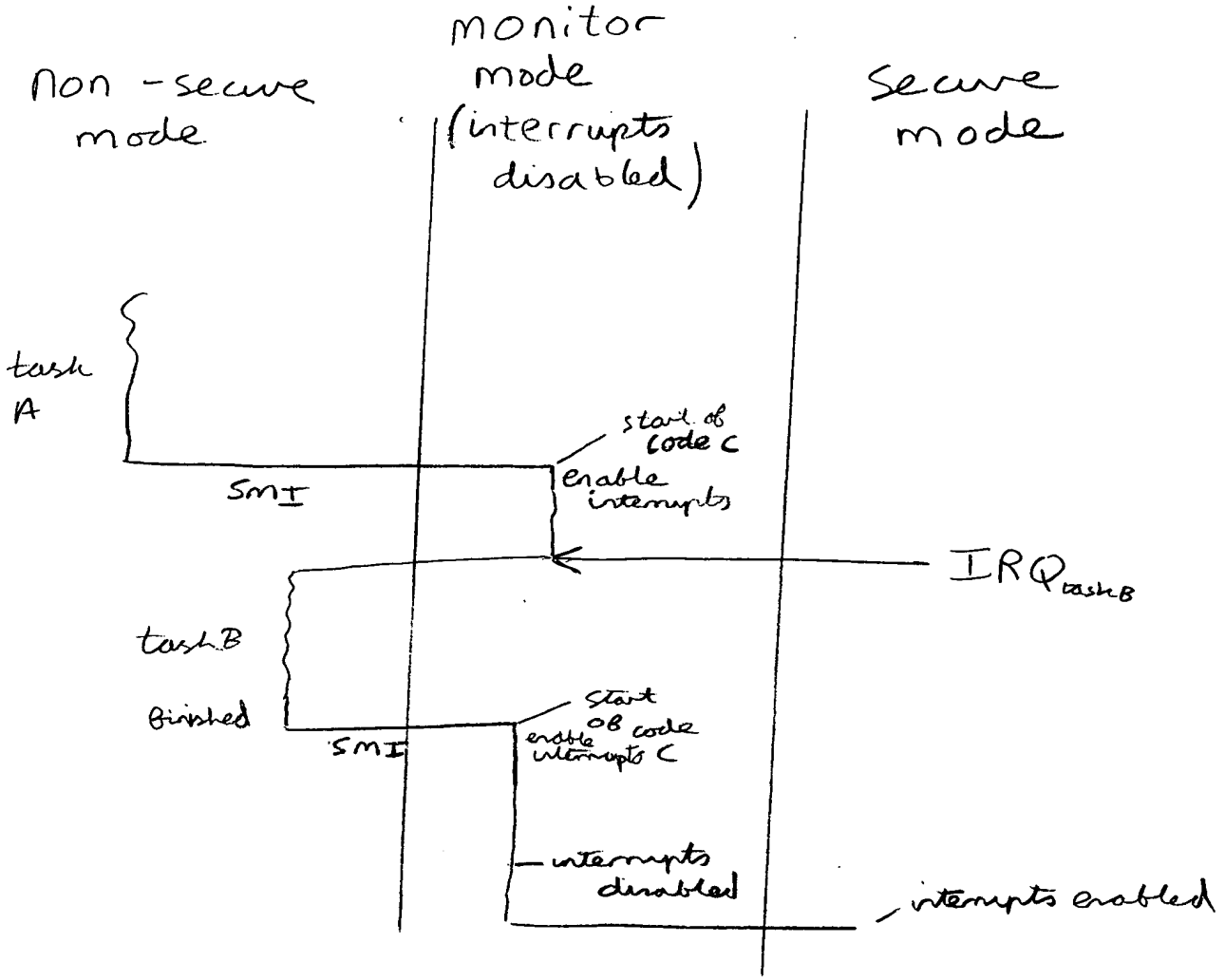

non-secure mode
monitor mode (interrupts disabled)
secure mode
task A
SMI
start of code C
enable interrupts
IRQ task B
task B
finished
SMI
start of code C
enable interrupts C
interrupts disabled
interrupts enabled

Fig. 18

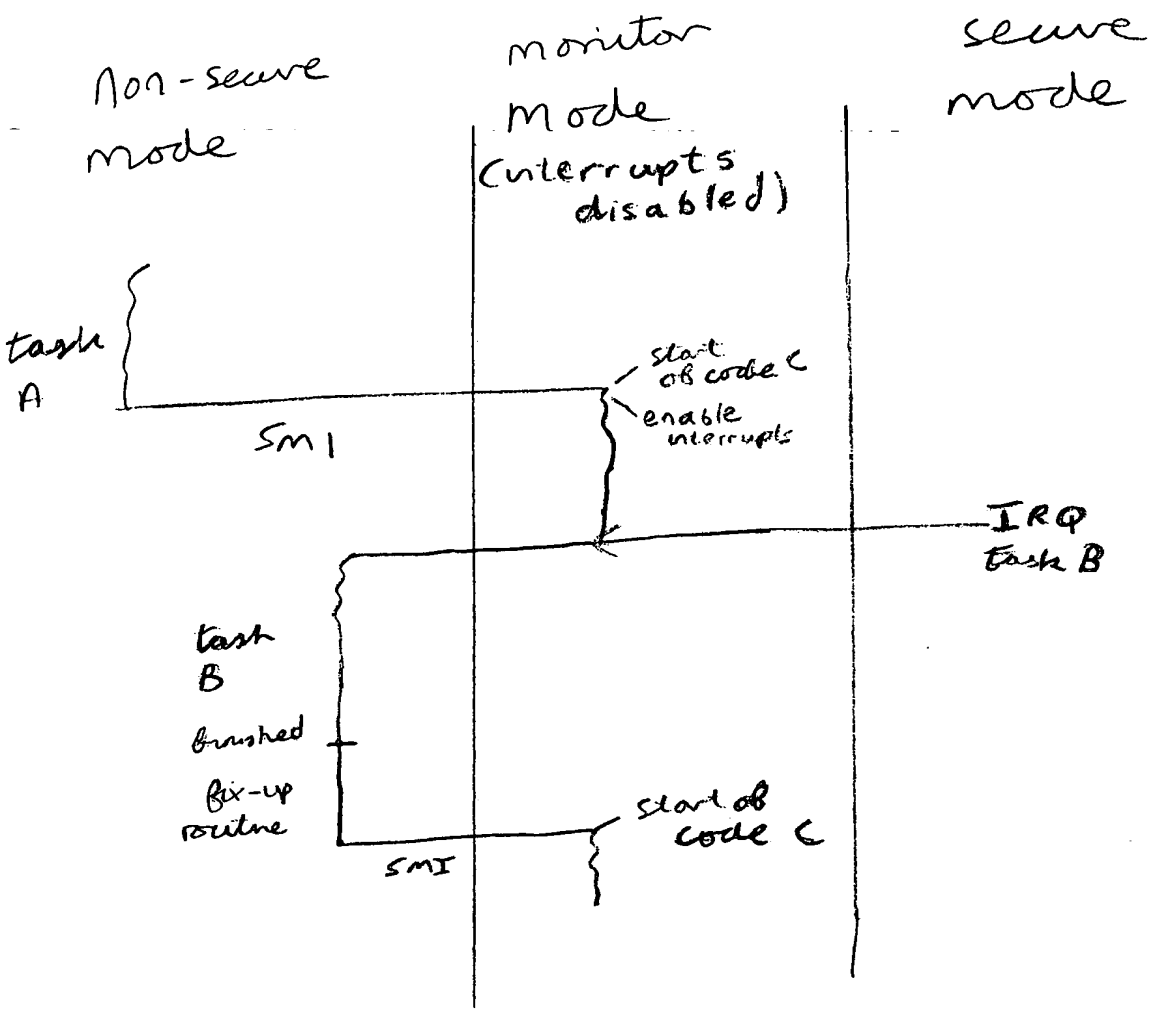

Non-secure mode
monitor mode (interrupts disabled)
secure mode
task A
SMI
start of code C
enable interrupts
IRQ task B
task B
flushed
fix-up routine
SMI
start of code C

Fig. 19

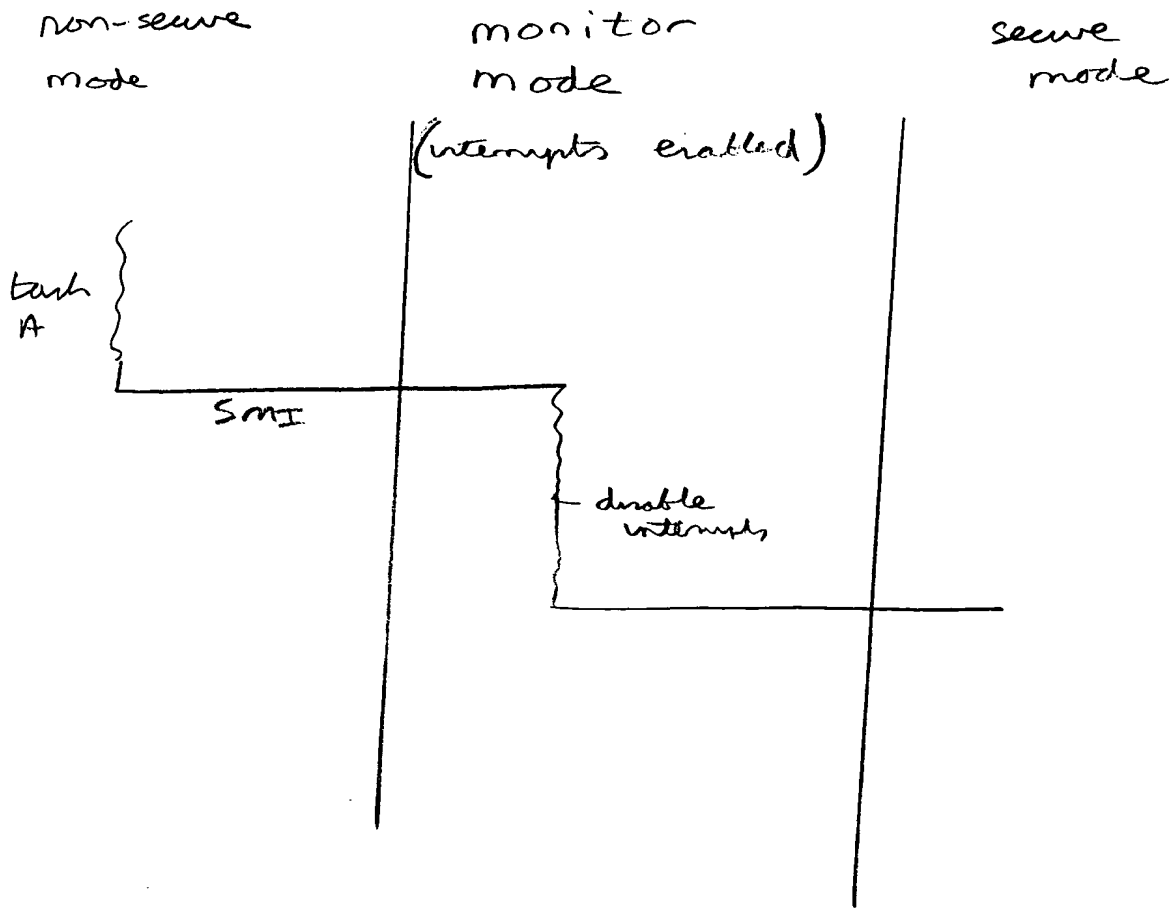

non-secure mode
monitor mode
(interrupts enabled)
secure mode
task A
SMI
disable interrupts

Fig. 20

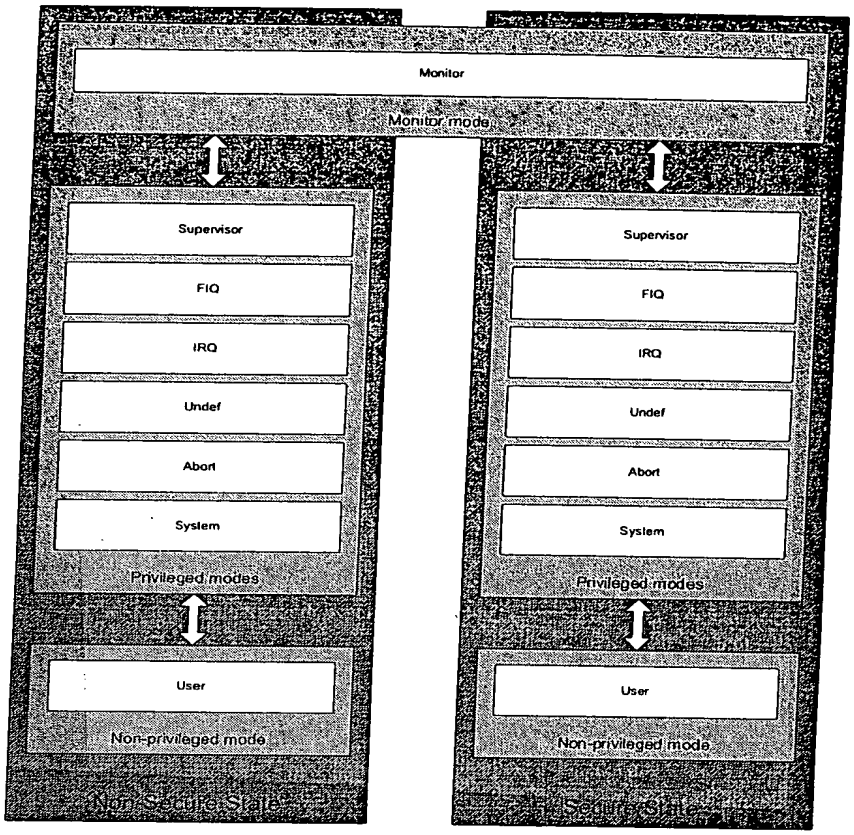

Monitor
Monitor mode
Supervisor
FIQ
IRQ
Undef
Abort
System
Privileged modes
User
Non-privileged mode
Non-Secure State
Supervisor
FIQ
IRQ
Undef
Abort
System
Privileged modes
User
Non-privileged mode
Secure State

FIGURE 21

User	System	Supervisor	Abort	Undefined	Interrupt	Fast Interrupt	*Monitor*
R0	R0	R0	R0	R0	R0	R0	R0
R1	R1	R1	R1	R1	R1	R1	R1
R2	R2	R2	R2	R2	R2	R2	R2
R3	R3	R3	R3	R3	R3	R3	R3
R4	R4	R4	R4	R4	R4	R4	R4
R5	R5	R5	R5	R5	R5	R5	R5
R6	R6	R6	R6	R6	R6	R6	R6
R7	R7	R7	R7	R7	R7	R7	R7
R8	R8	R8	R8	R8	R8	R8_fiq	R8
R9	R9	R9	R9	R9	R9	R9_fiq	R9
R10	R10	R10	R10	R10	R10	R10_fiq	R10
R11	R11	R11	R11	R11	R11	R11_fiq	R11
R12	R12	R12	R12	R12	R12	R12_fiq	R12
R13	R13	~~R13_svc~~	~~R13_abt~~	R13_und	R13_irq	R13_fiq	*R13_mon*
R14	R14	~~R14_svc~~	~~R14_sbt~~	R14_und	R14_irq	R14_fiq	*R14_mon*
PC	PC	PC	PC	PC	PC	PC	*PC*

CPSR	CPSR	CPSR	CPSR	CPSR	CPSR	CPSR	*CPSR*
		SPSR_svc	SPSR_abt	SPSR_und	SPSR_irq	SPSR_fiq	*SPSR_mon*

FIGURE 22

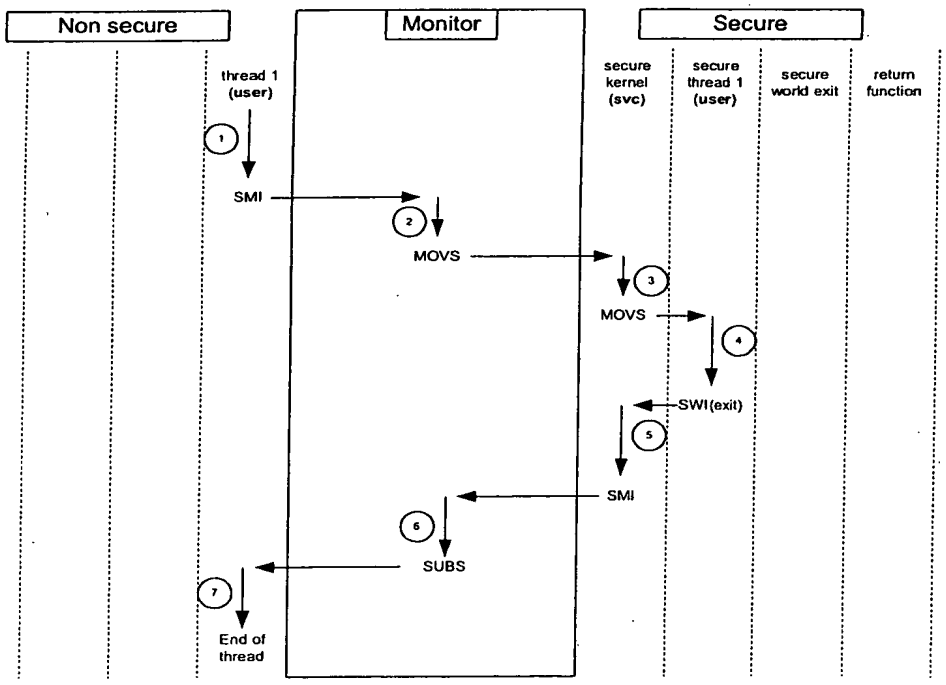
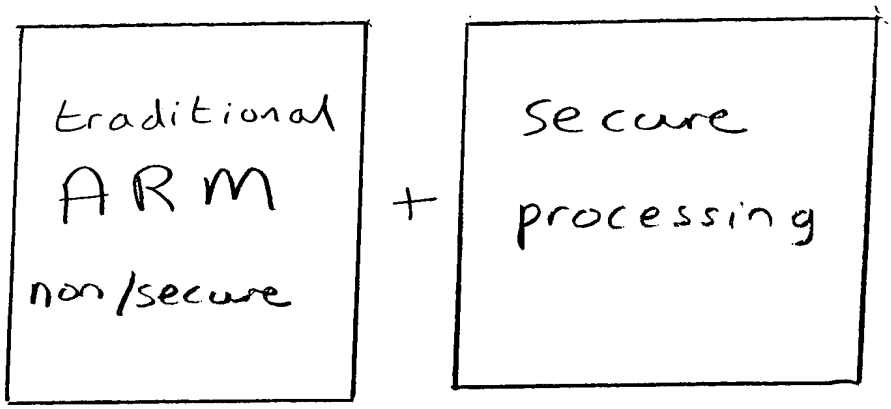
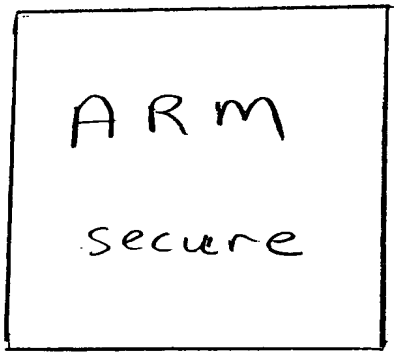


FIGURE 23





S = 1

Fig. 24

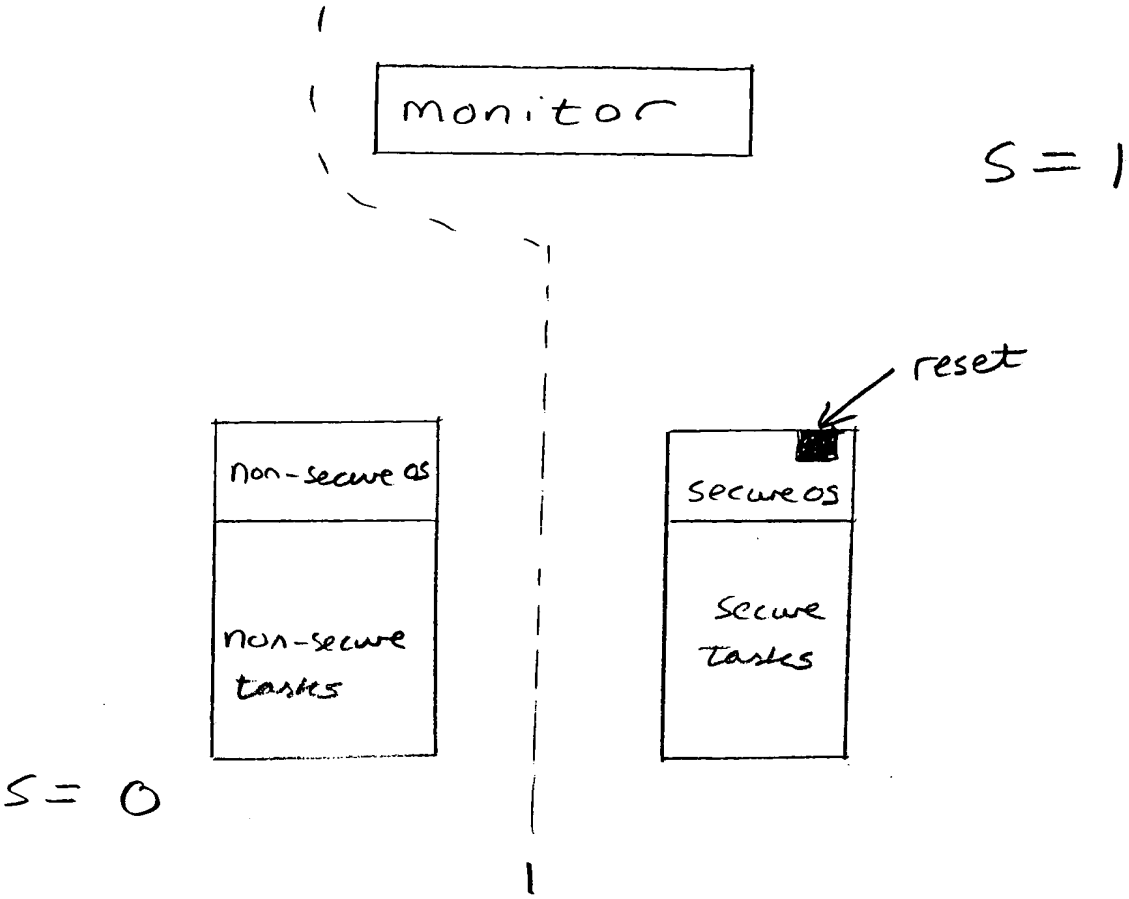

monitor
S = 1
reset
non-secure os
non-secure tasks
secure os
secure tasks
S = 0

Fig. 25

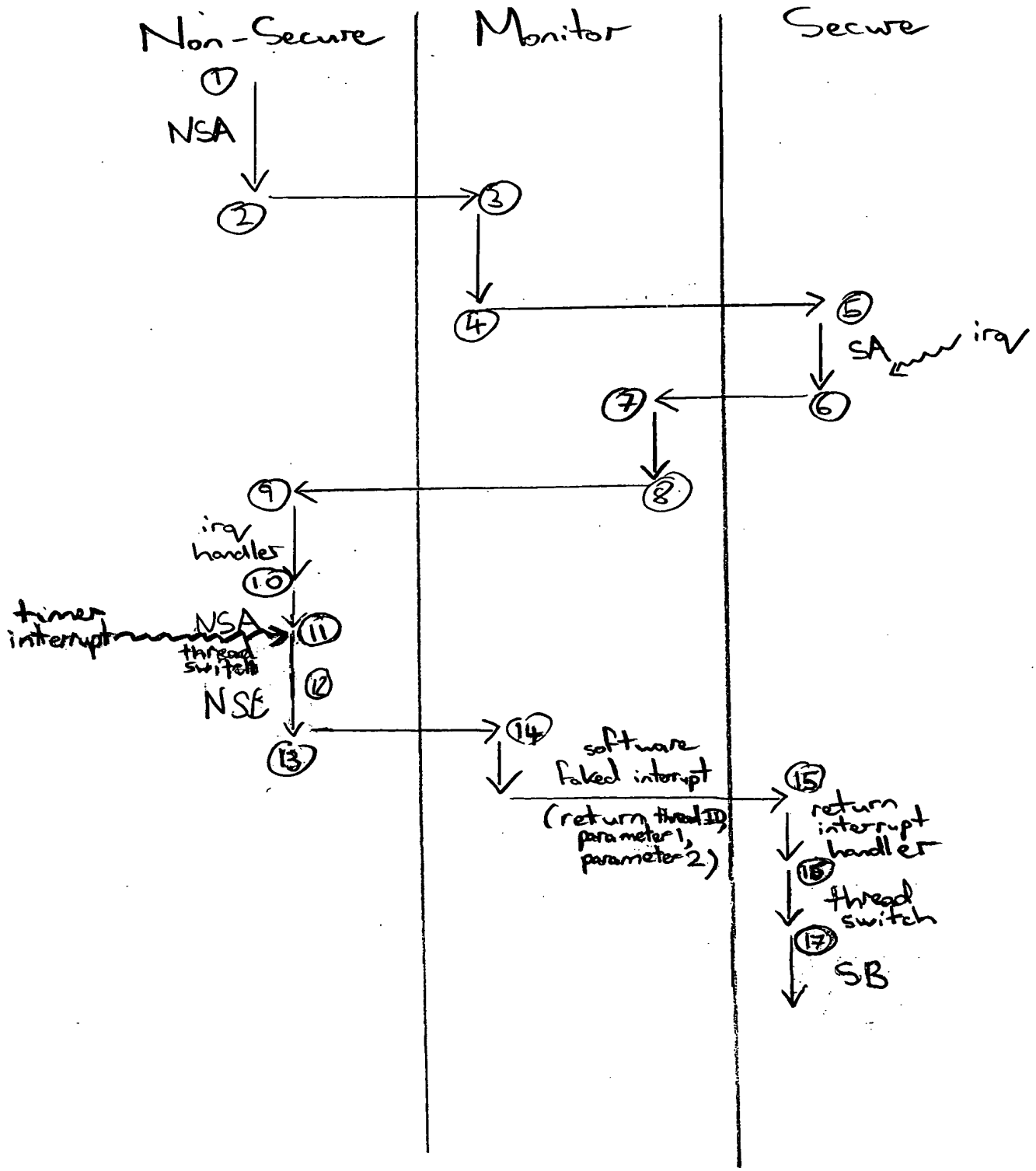

Non-Secure
Monitor
Secure
1
NSA
2
3
4
5
SA
irq
6
7
8
9
irq handler
10
timer interrupt
NSA
thread switch
NSE
11
12
13
14
software
faked interrupt
(return, thread ID, parameter 1, parameter 2)
15
return interrupt handler
16
thread switch
17
SB

Fig. 26

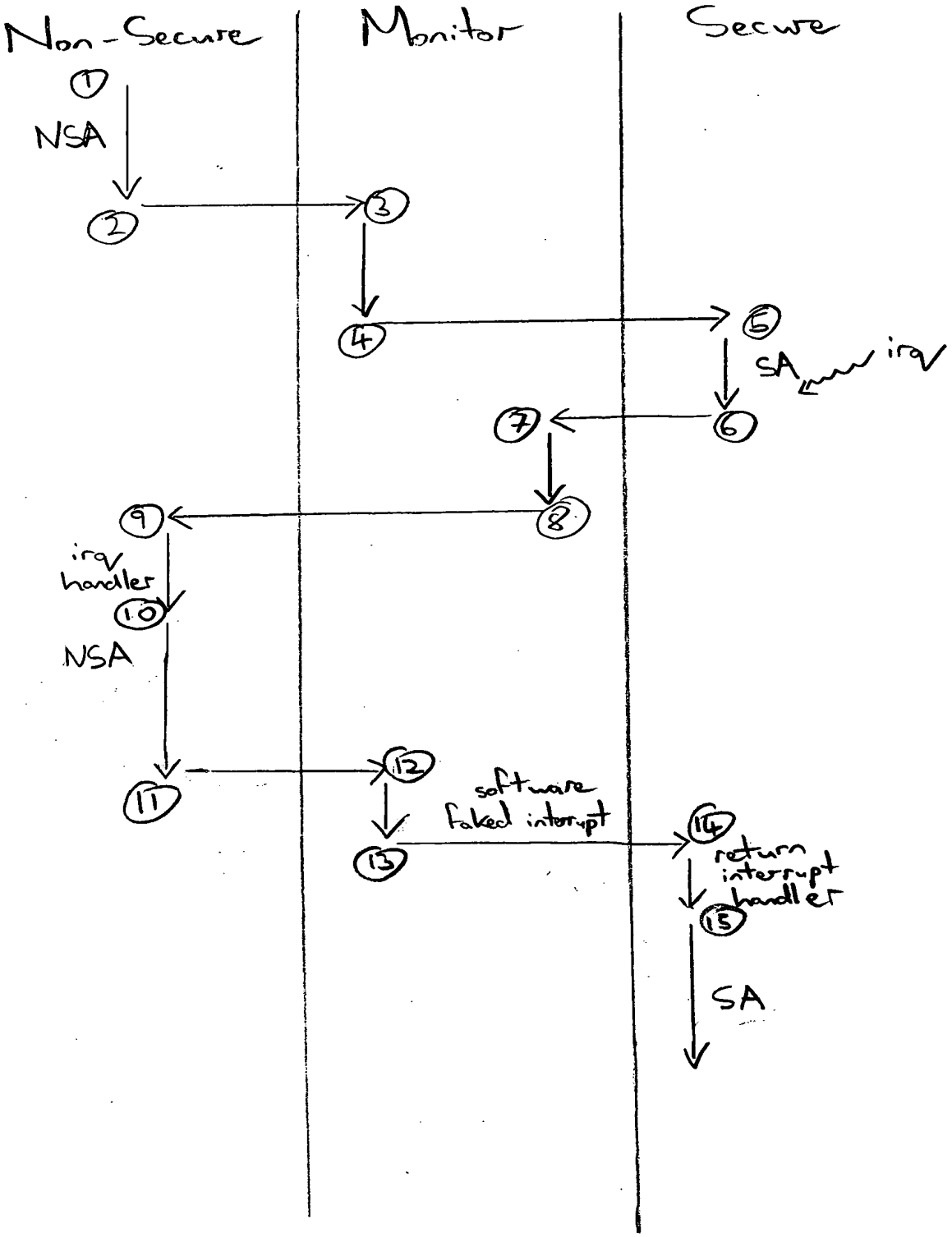

Non-Secure
Monitor
Secure
1
NSA
2
3
4
5
SA
irq
6
7
8
9
irq handler
10
NSA
11
12
software faked interrupt
13
14
return interrupt handler
15
SA

Fig. 27

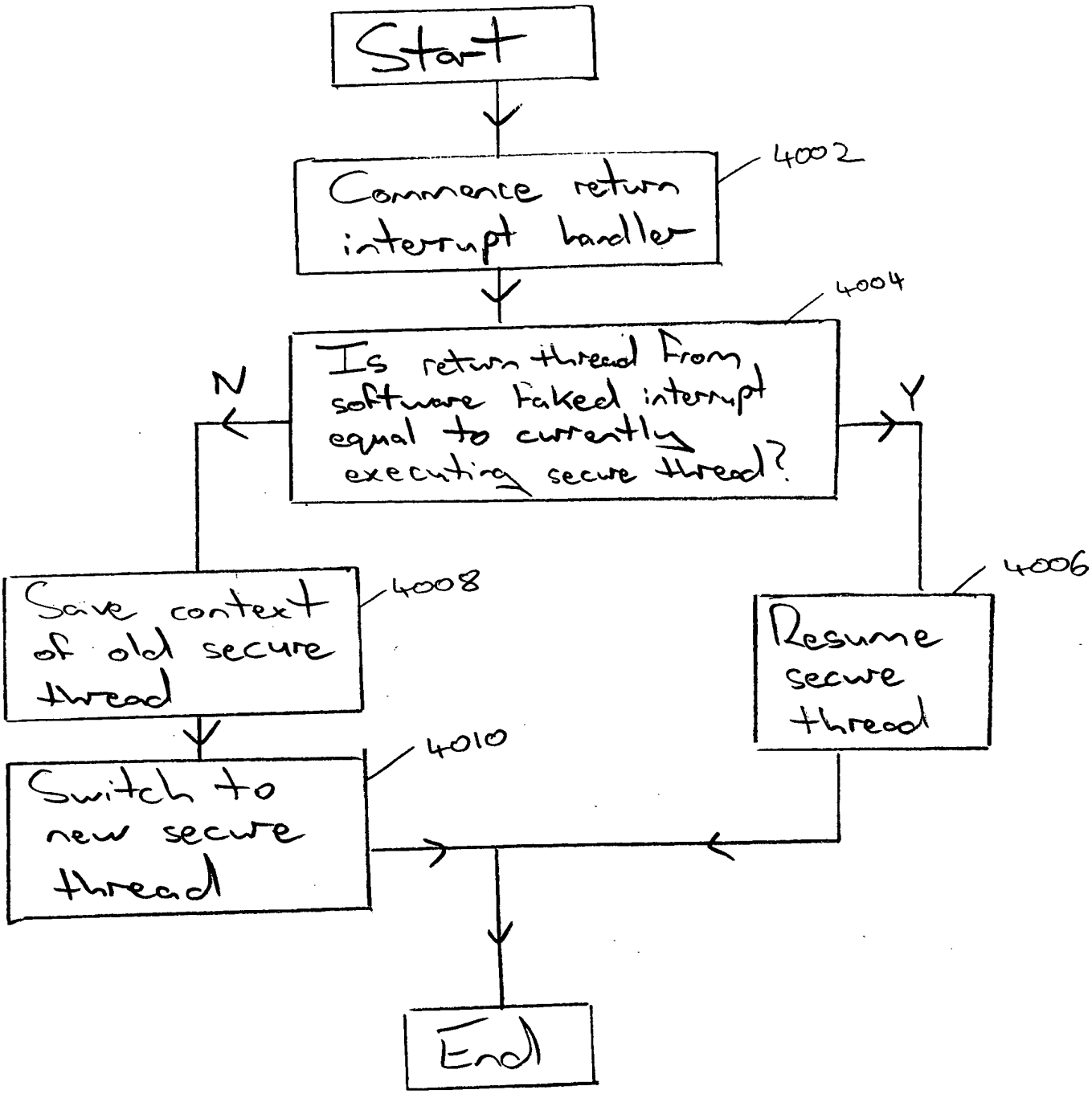

Start
Commence return interrupt handler
4002
Is return thread from software faked interrupt equal to currently executing secure thread?
4004
N
Y
Save context of old secure thread
4008
Resume secure thread
4006
Switch to new secure thread
4010
End

Fig. 28

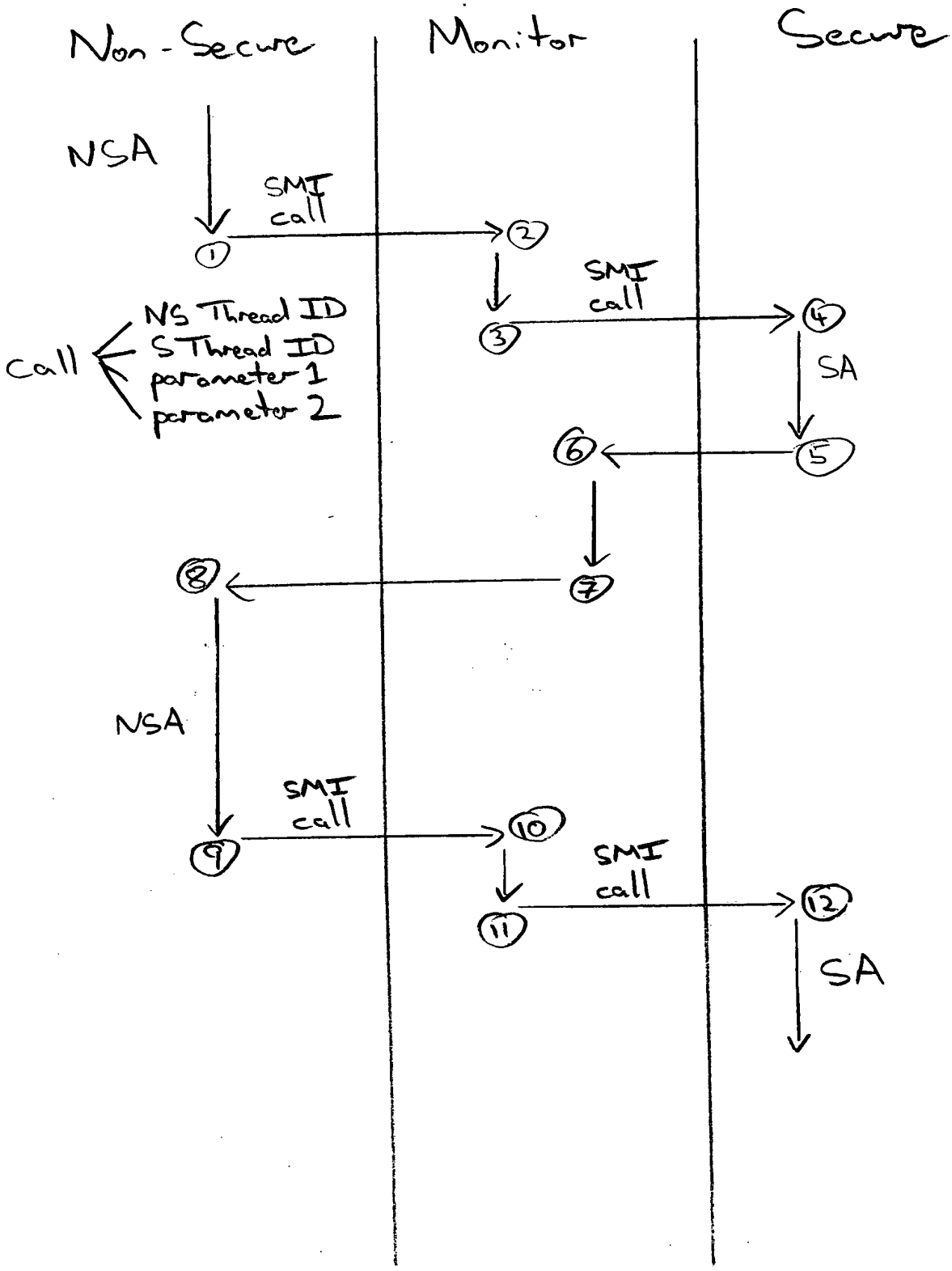

Non-Secure
Monitor
Secure
NSA
SMI call
1
2
3
SMI call
4
SA
5
6
7
8
NSA
SMI call
9
10
11
SMI call
12
SA
Call
NS Thread ID
S Thread ID
parameter 1
parameter 2

Fig. 29

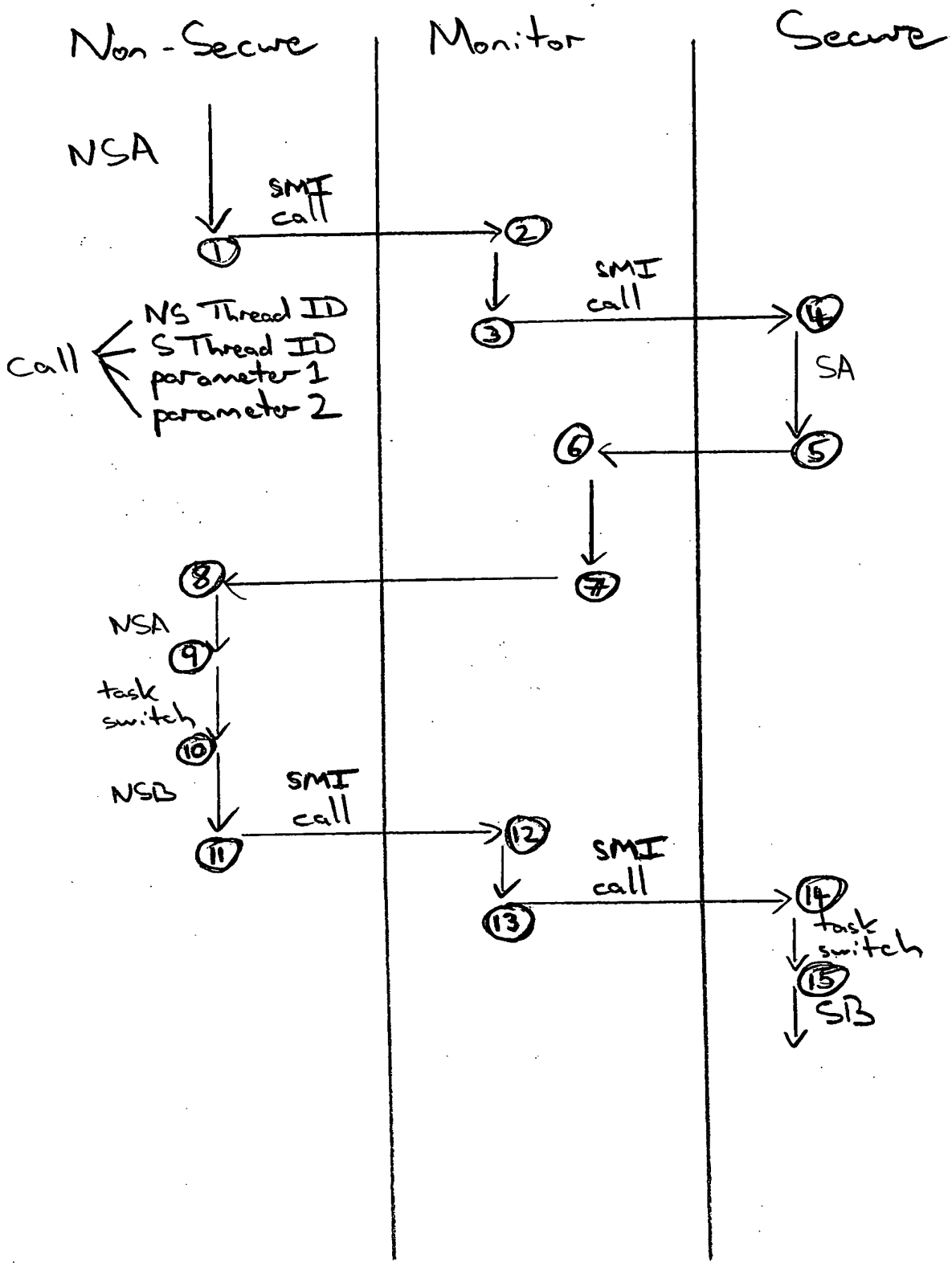

Non-Secure
Monitor
Secure
NSA
SMI call
1
2
SMI call
3
4
SA
5
6
7
8
NS Thread ID
S Thread ID
parameter 1
parameter 2
Call
NSA
9
task switch
10
NSB
SMI call
11
12
SMI call
13
14
task switch
15
SB

Fig. 30

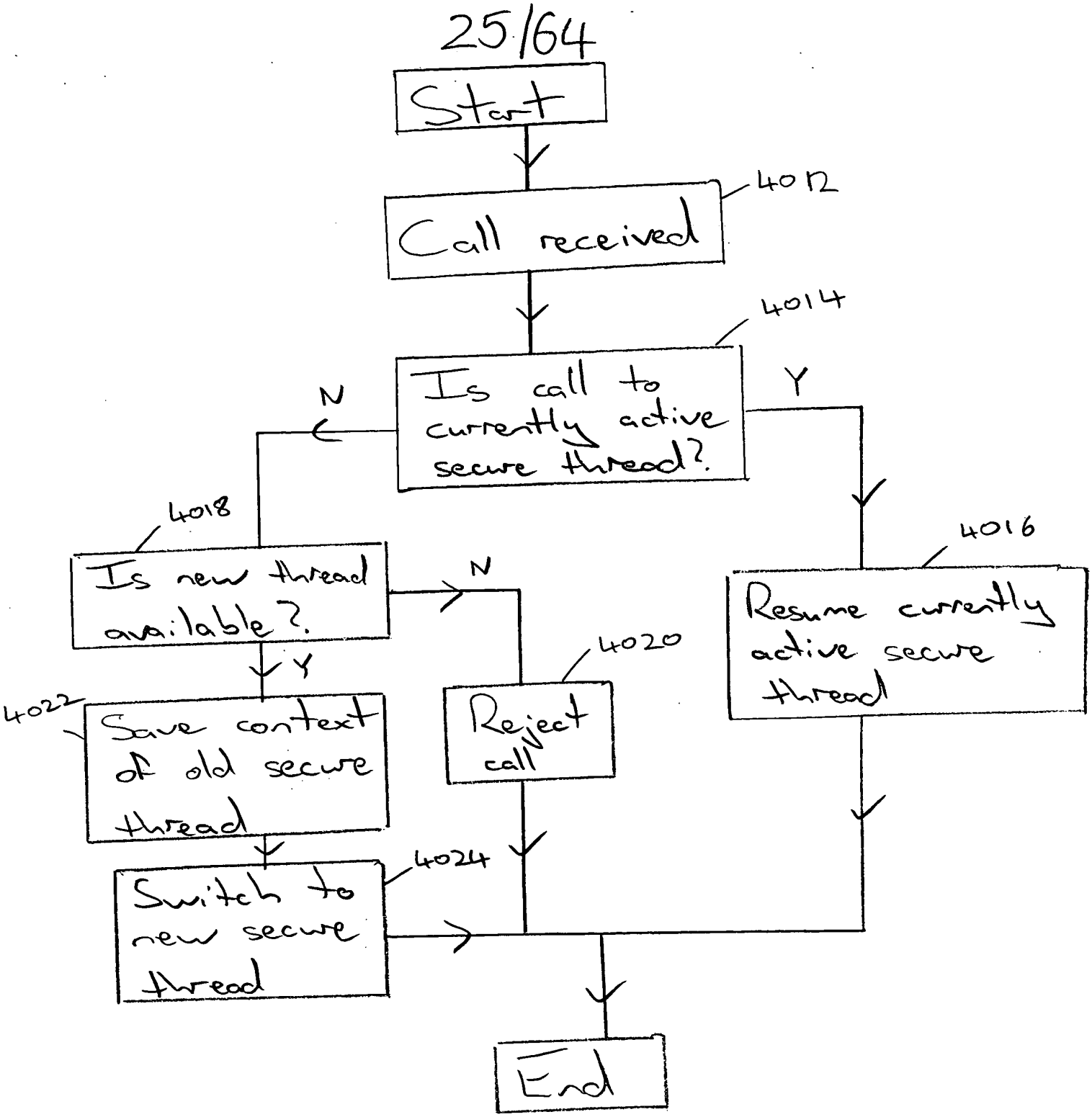

Start
Call received
4012
Is call to currently active secure thread?
4014
N
Y
Is new thread available?
4018
N
Y
Resume currently active secure thread
4016
Save context of old secure thread
4022
Reject call
4020
Switch to new secure thread
4024
End

Fig. 31

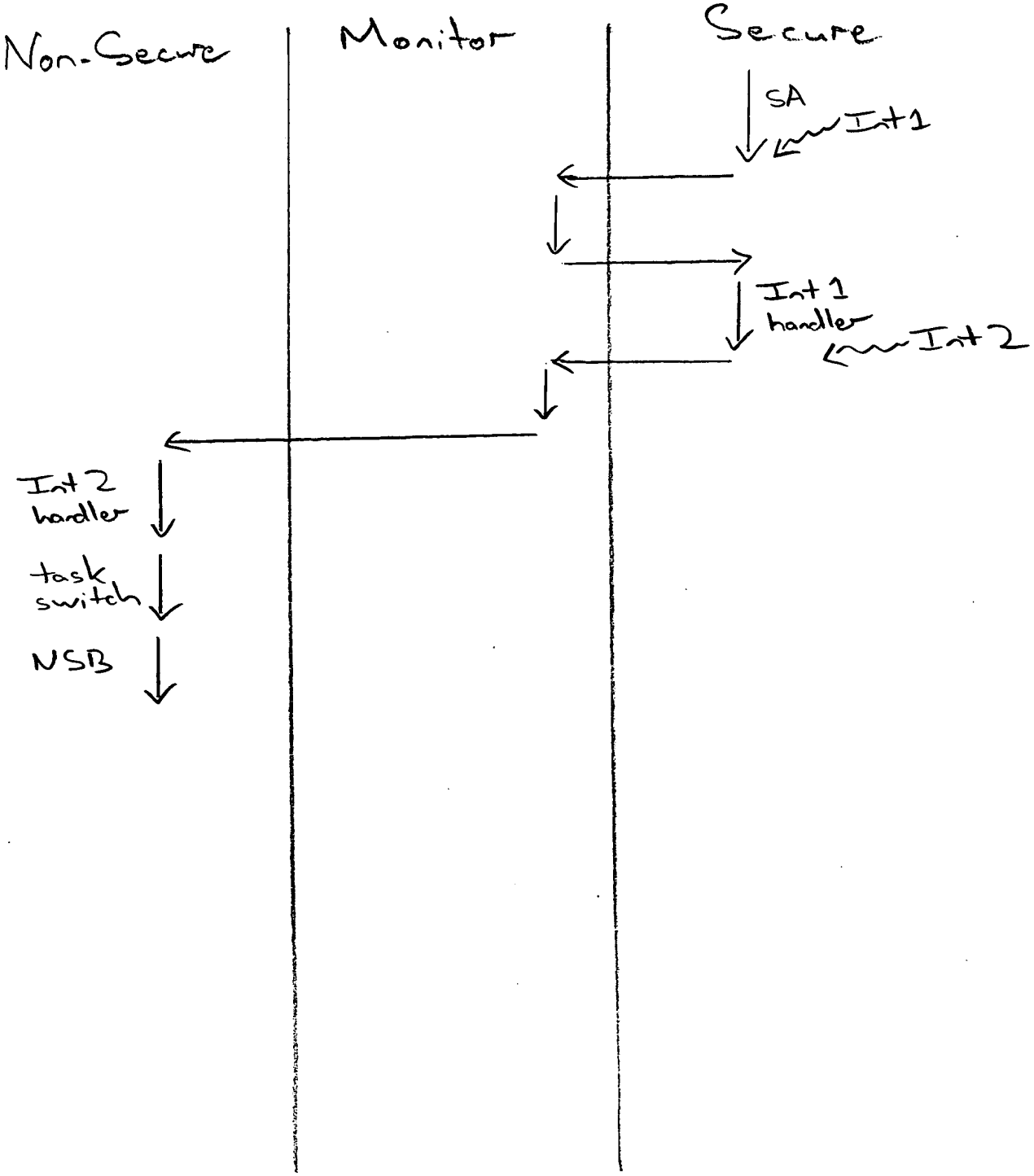

Non-Secure
Monitor
Secure
SA
Int 1
Int 1 handler
Int 2
Int 2 handler
task switch
NSB

Fig. 32

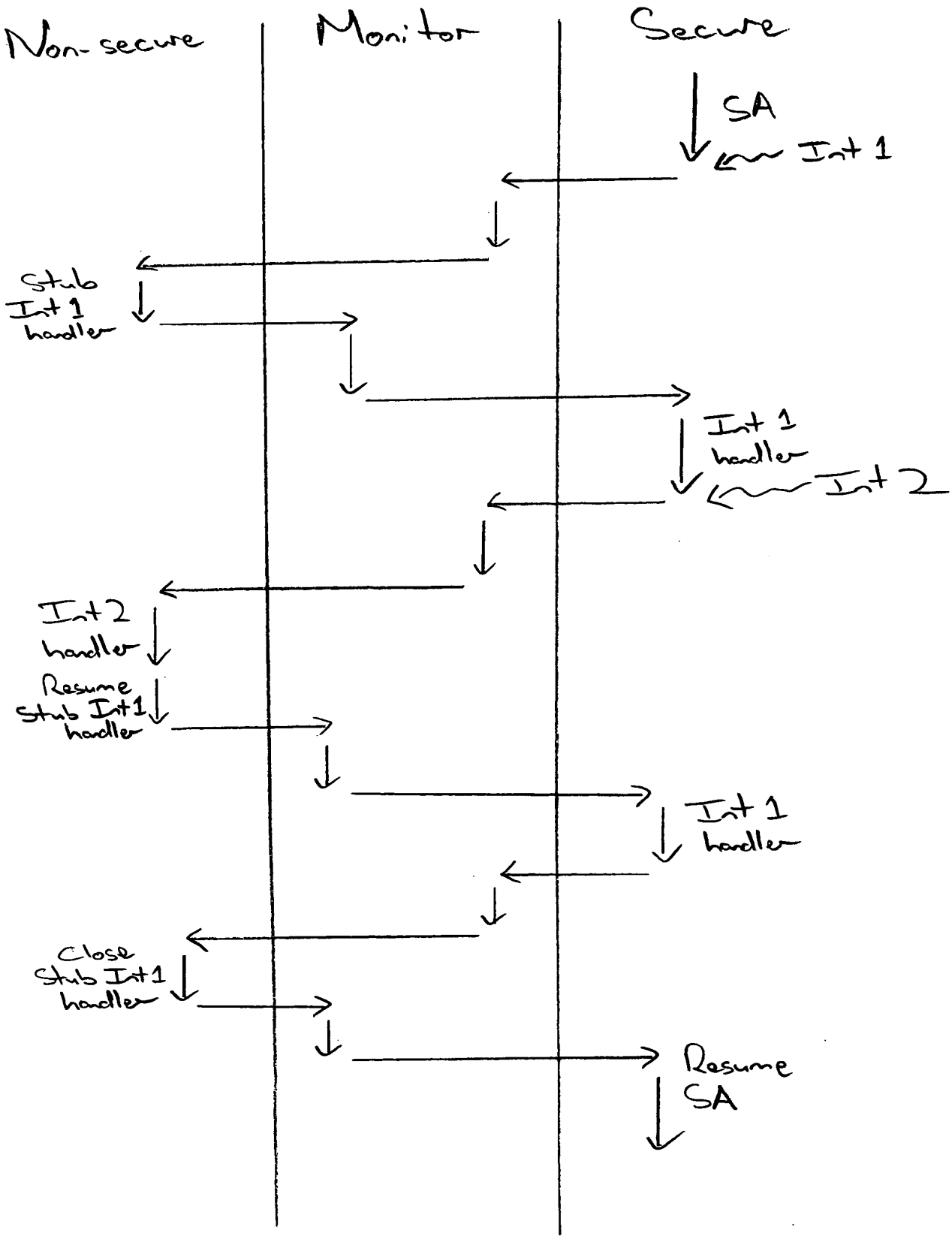

Non-secure
Monitor
Secure
SA
Int 1
Stub Int 1 handler
Int 1 handler
Int 2
Int 2 handler
Resume Stub Int1 handler
Int 1 handler
Close Stub Int1 handler
Resume SA

Fig 33

Interrupt Type / Priority	How Handled
1	S
2	S
3	NS
4	NS/S
5	NS
6	NS/S
7	NS
⋮	⋮

no S only handlers lower than highes NS handler

Fig. 34

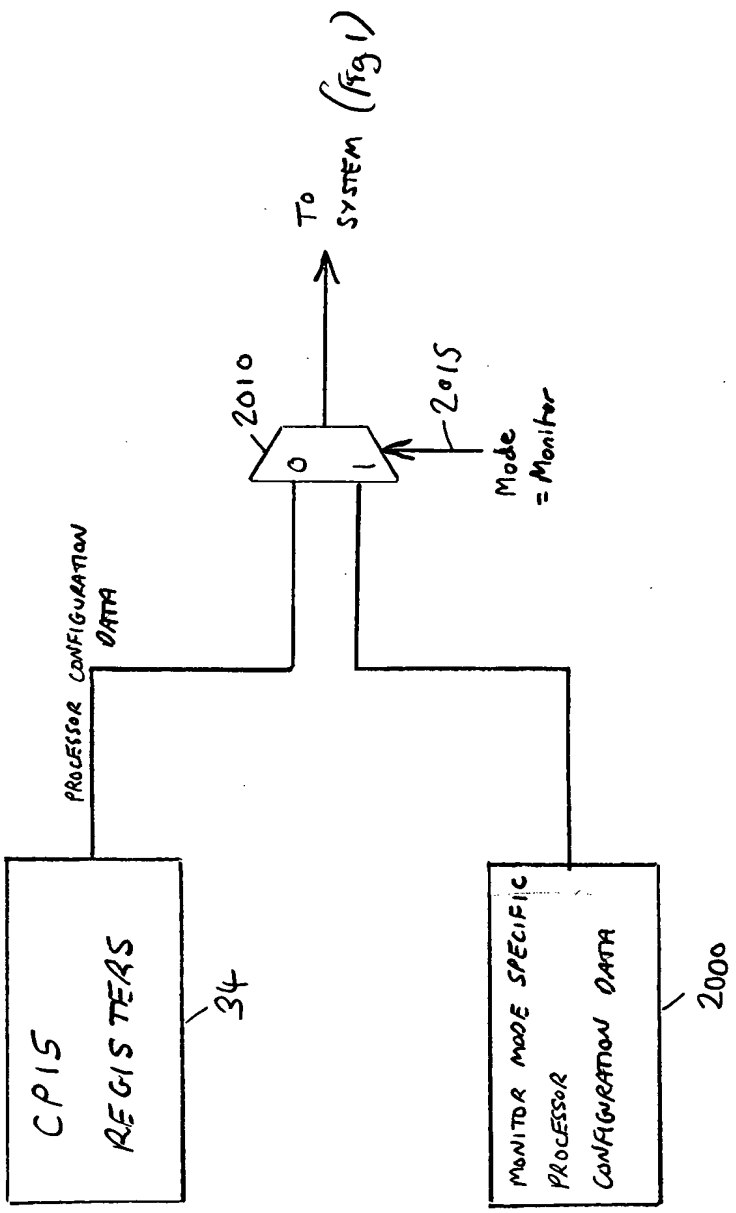

CP15
REGISTERS
34
PROCESSOR CONFIGURATION DATA
MONITOR MODE SPECIFIC
PROCESSOR
CONFIGURATION DATA
2000
2010
0
1
2015
Mode
= Monitor
To
SYSTEM (Fig 1)

FIG. 35

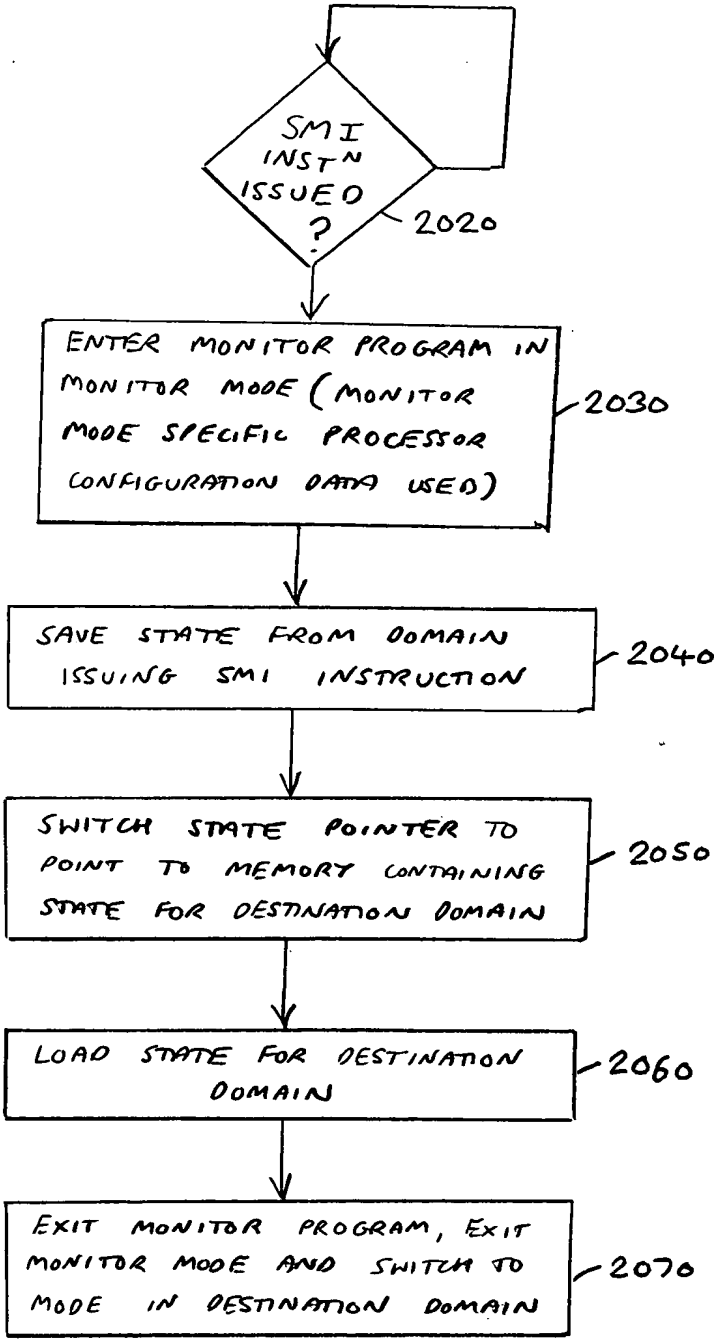

SMI INSTN ISSUED ?
2020
ENTER MONITOR PROGRAM IN MONITOR MODE (MONITOR MODE SPECIFIC PROCESSOR CONFIGURATION DATA USED)
2030
SAVE STATE FROM DOMAIN ISSUING SMI INSTRUCTION
2040
SWITCH STATE POINTER TO POINT TO MEMORY CONTAINING STATE FOR DESTINATION DOMAIN
2050
LOAD STATE FOR DESTINATION DOMAIN
2060
EXIT MONITOR PROGRAM, EXIT MONITOR MODE AND SWITCH TO MODE IN DESTINATION DOMAIN
2070

FIG. 36

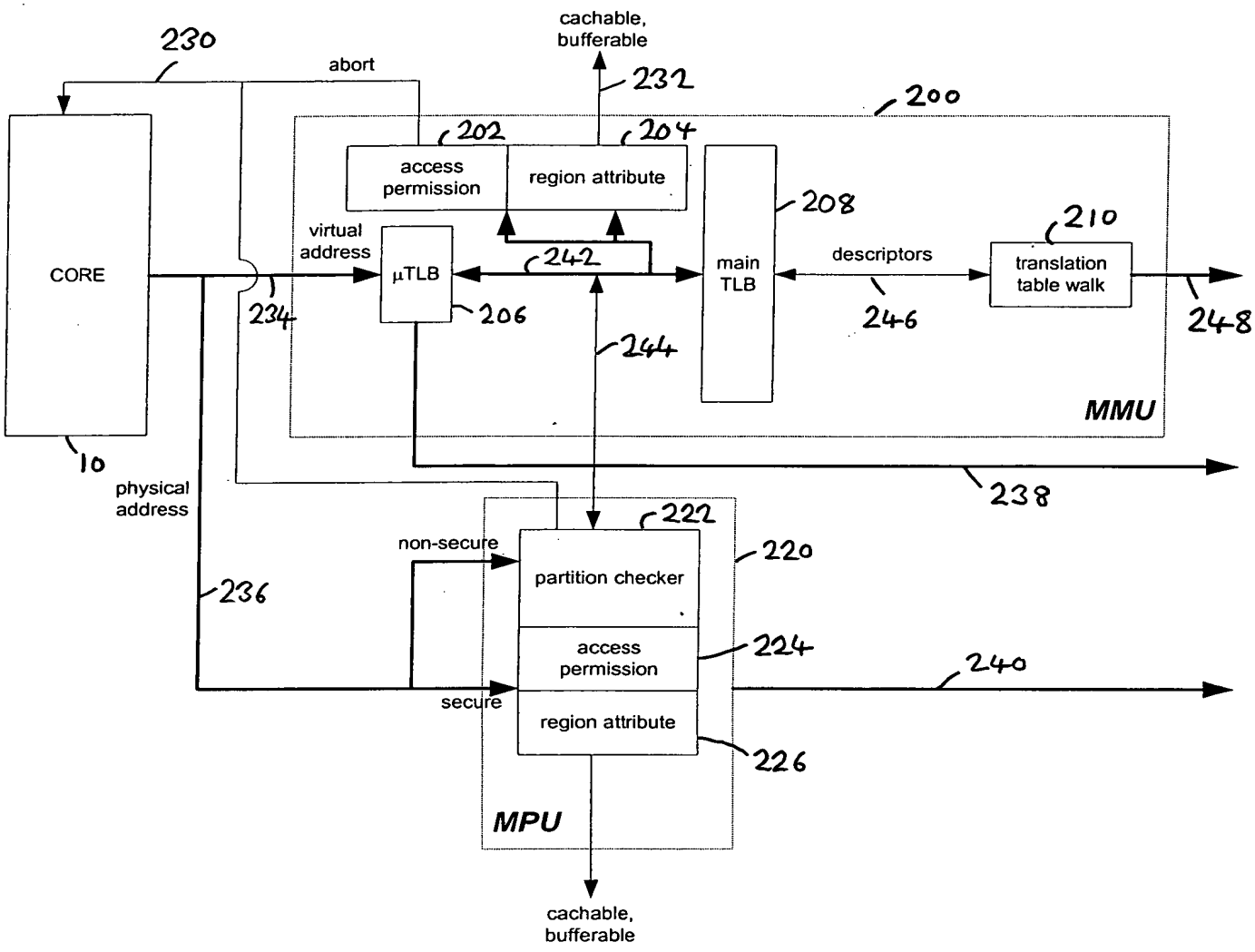

230
abort
cachable, bufferable
232
200
202
204
access permission
region attribute
208
210
virtual address
μTLB
242
main TLB
descriptors
translation table walk
CORE
234
206
246
248
244
MMU
10
physical address
238
222
non-secure
partition checker
220
236
access permission
224
240
secure
region attribute
226
MPU
cachable, bufferable

FIG. 37

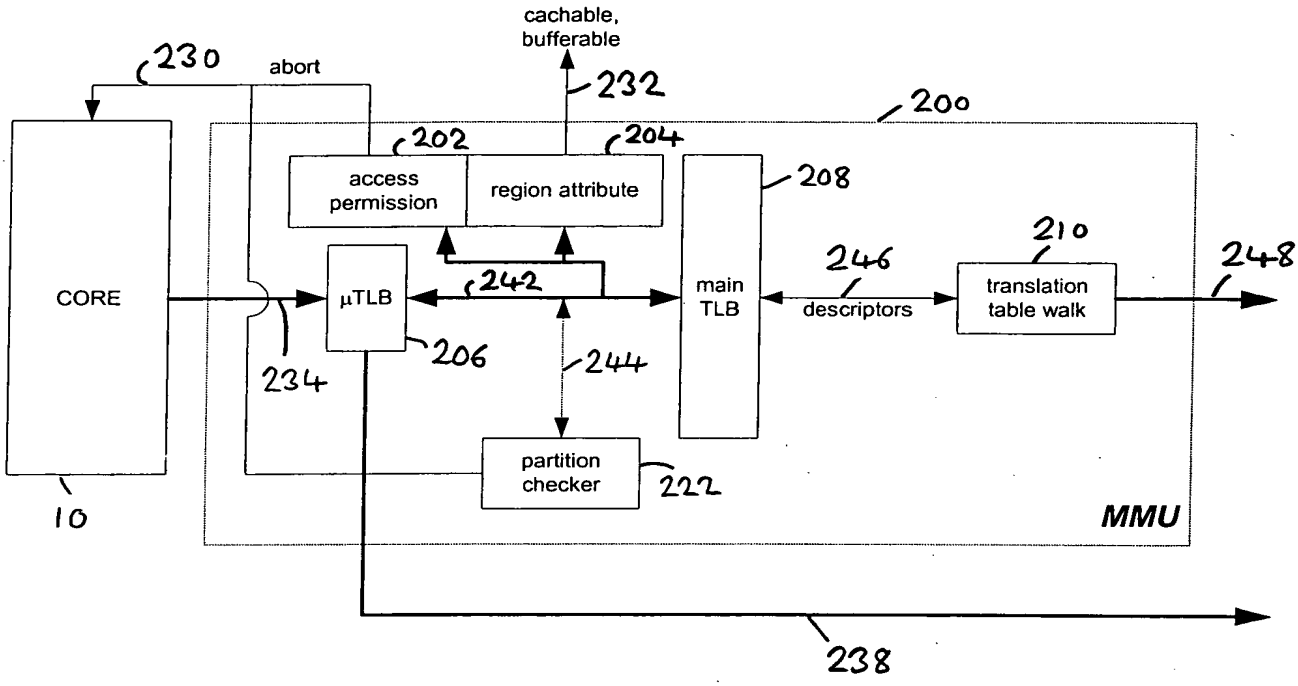

cachable,
bufferable
230
abort
232
200
202
204
access
permission
region attribute
208
CORE
μTLB
242
main
TLB
246
descriptors
210
translation
table walk
248
234
206
244
partition
checker
222
MMU
10
238

FIG. 38

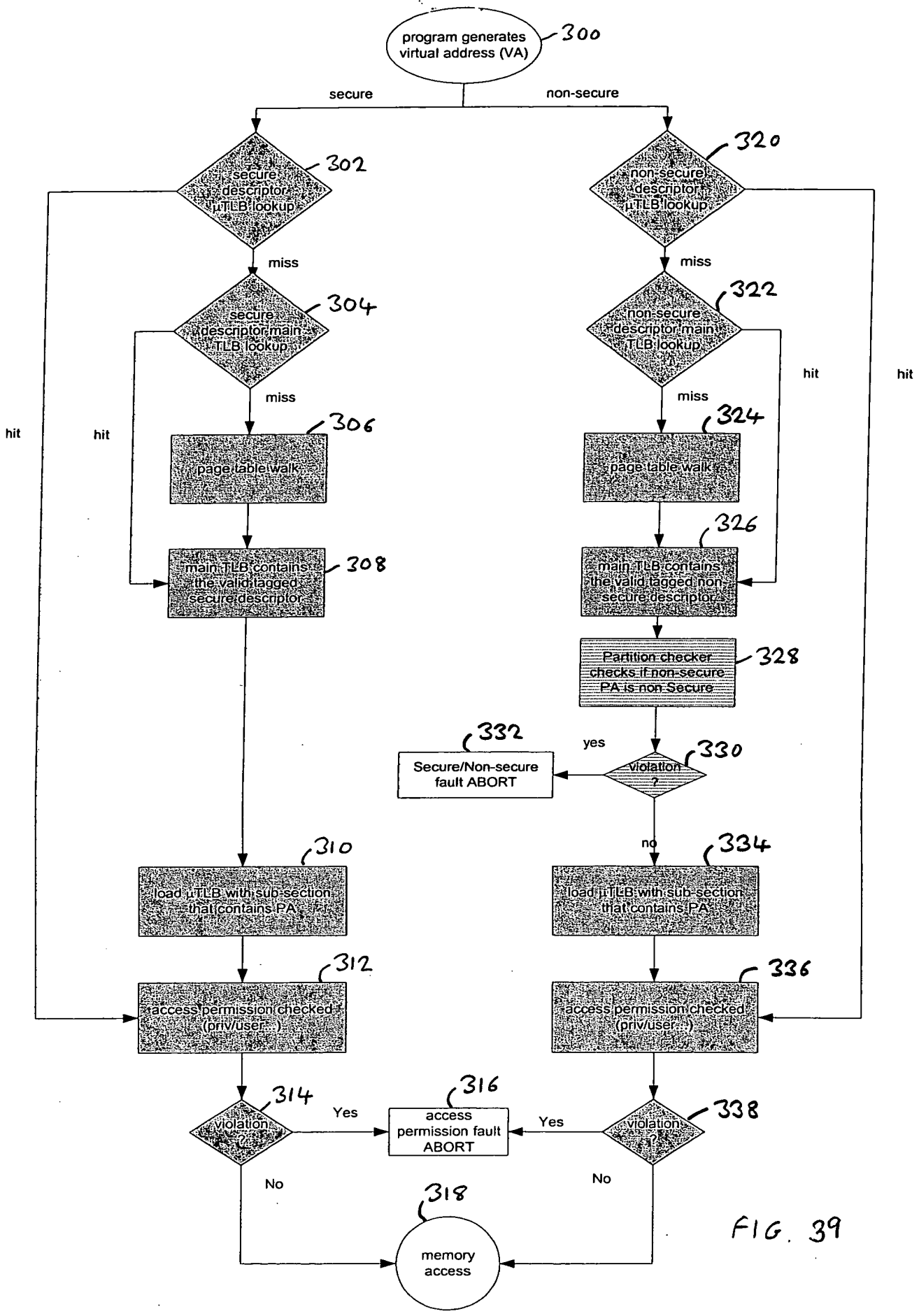

program generates virtual address (VA)
300
secure
non-secure
secure descriptor µTLB lookup
302
miss
secure descriptor main TLB lookup
304
miss
page table walk
306
main TLB contains the valid tagged secure descriptor
308
hit
hit
load µTLB with sub-section that contains PA
310
access permission checked (priv/user...)
312
violation ?
314
Yes
No
non-secure descriptor µTLB lookup
320
miss
non-secure descriptor main TLB lookup
322
miss
page table walk
324
main TLB contains the valid tagged non-secure descriptor
326
hit
hit
Partition checker checks if non-secure PA is non-Secure
328
yes
violation ?
330
Secure/Non-secure fault ABORT
332
no
load µTLB with sub-section that contains PA
334
access permission checked (priv/user...)
336
violation ?
338
Yes
No
access permission fault ABORT
316
memory access
318

FIG. 39

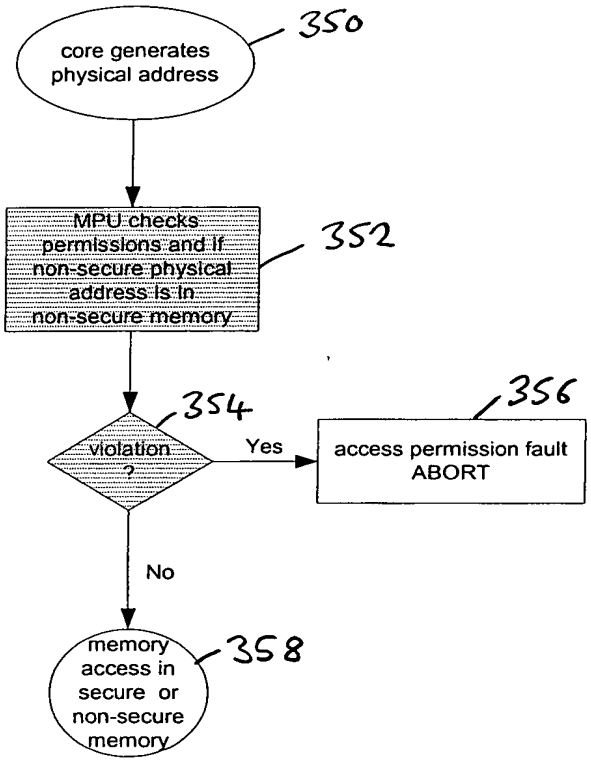

core generates physical address
350
MPU checks permissions and if non-secure physical address is in non-secure memory
352
354
violation ?
Yes
356
access permission fault ABORT
No
memory access in secure or non-secure memory
358

FIG. 40

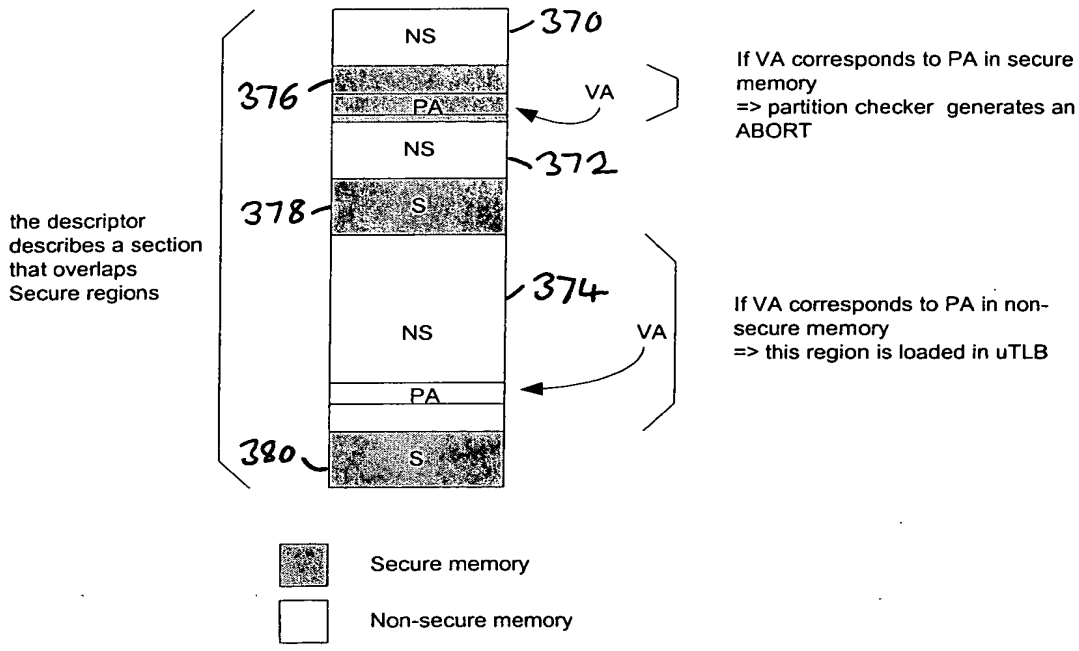

NS
370
376
PA
VA
NS
372
378
S
374
NS
VA
PA
380
S
the descriptor describes a section that overlaps Secure regions
If VA corresponds to PA in secure memory
=> partition checker generates an ABORT
If VA corresponds to PA in non-secure memory
=> this region is loaded in uTLB
Secure memory
Non-secure memory

FIG. 41

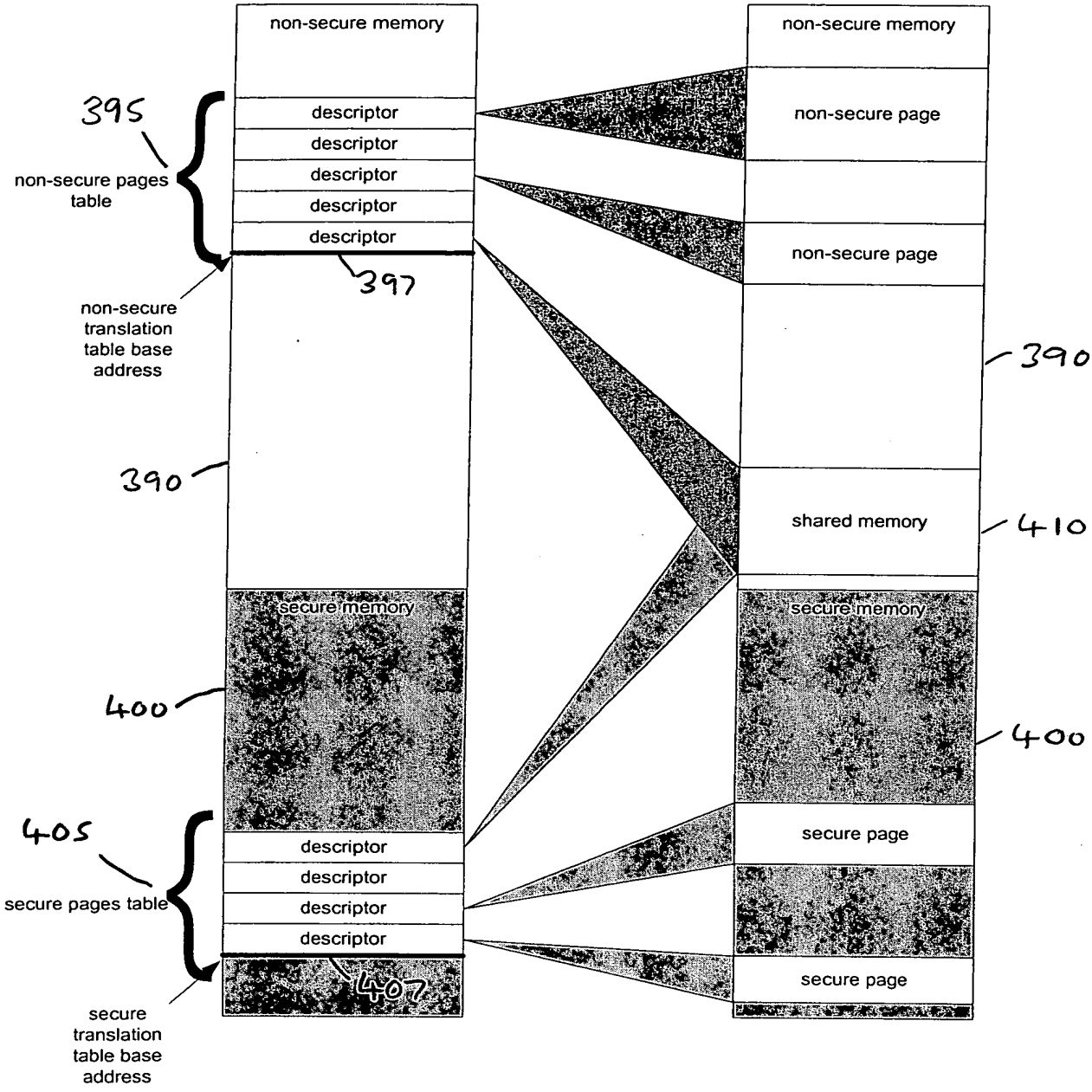

non-secure memory
395
descriptor
descriptor
descriptor
descriptor
descriptor
non-secure pages table
397
non-secure translation table base address
390
secure memory
400
405
descriptor
descriptor
descriptor
descriptor
secure pages table
407
secure translation table base address
non-secure memory
non-secure page
non-secure page
390
shared memory
410
secure memory
400
secure page
secure page

FIG. 42

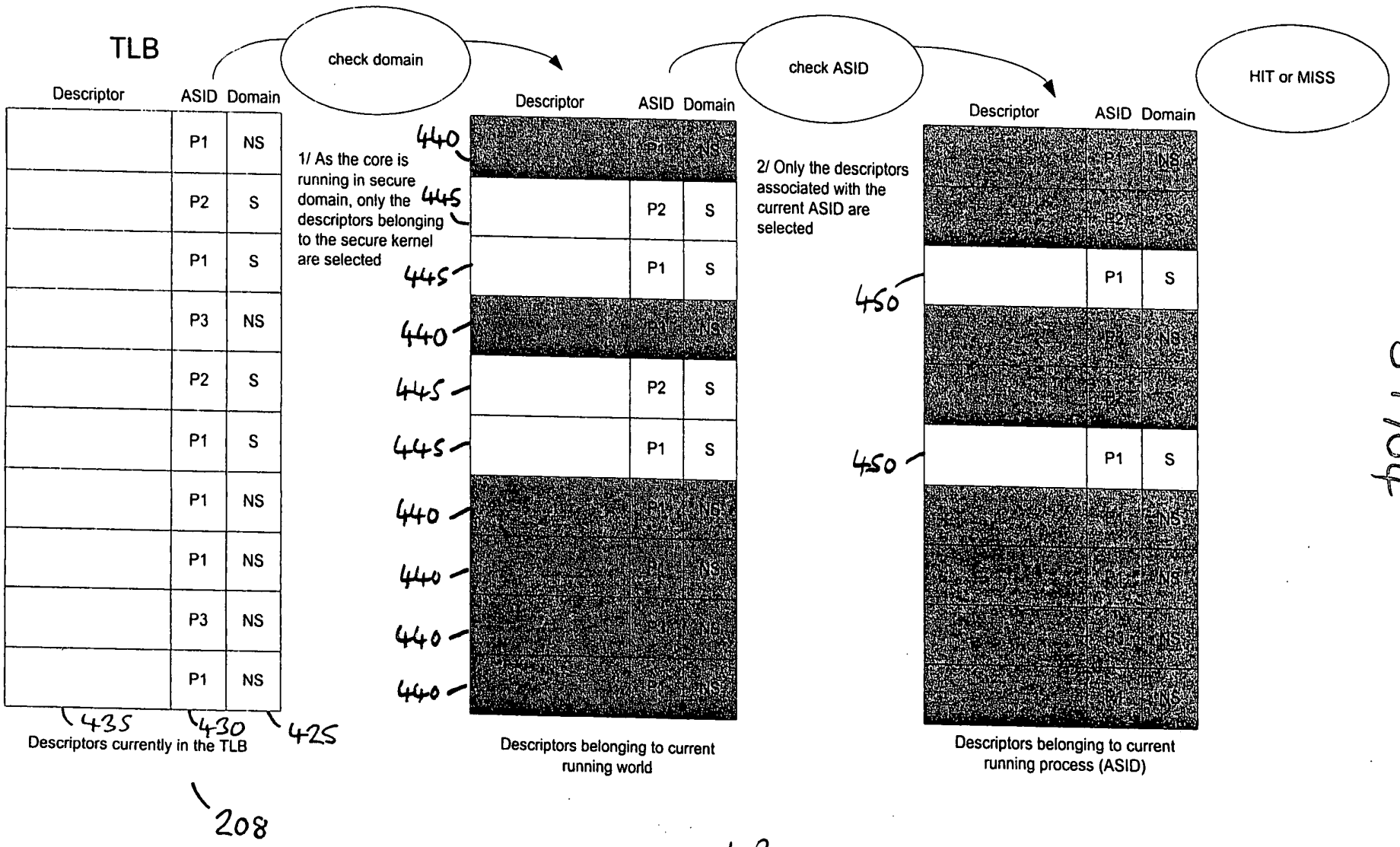

TLB
Descriptor
ASID
Domain
P1 NS
P2 S
P1 S
P3 NS
P2 S
P1 S
P1 NS
P1 NS
P3 NS
P1 NS
435
430
425
Descriptors currently in the TLB
208
check domain
1/ As the core is running in secure domain, only the descriptors belonging to the secure kernel are selected
440
445
Descriptors belonging to current running world
check ASID
2/ Only the descriptors associated with the current ASID are selected
450
Descriptors belonging to current running process (ASID)
HIT or MISS

FIG. 43

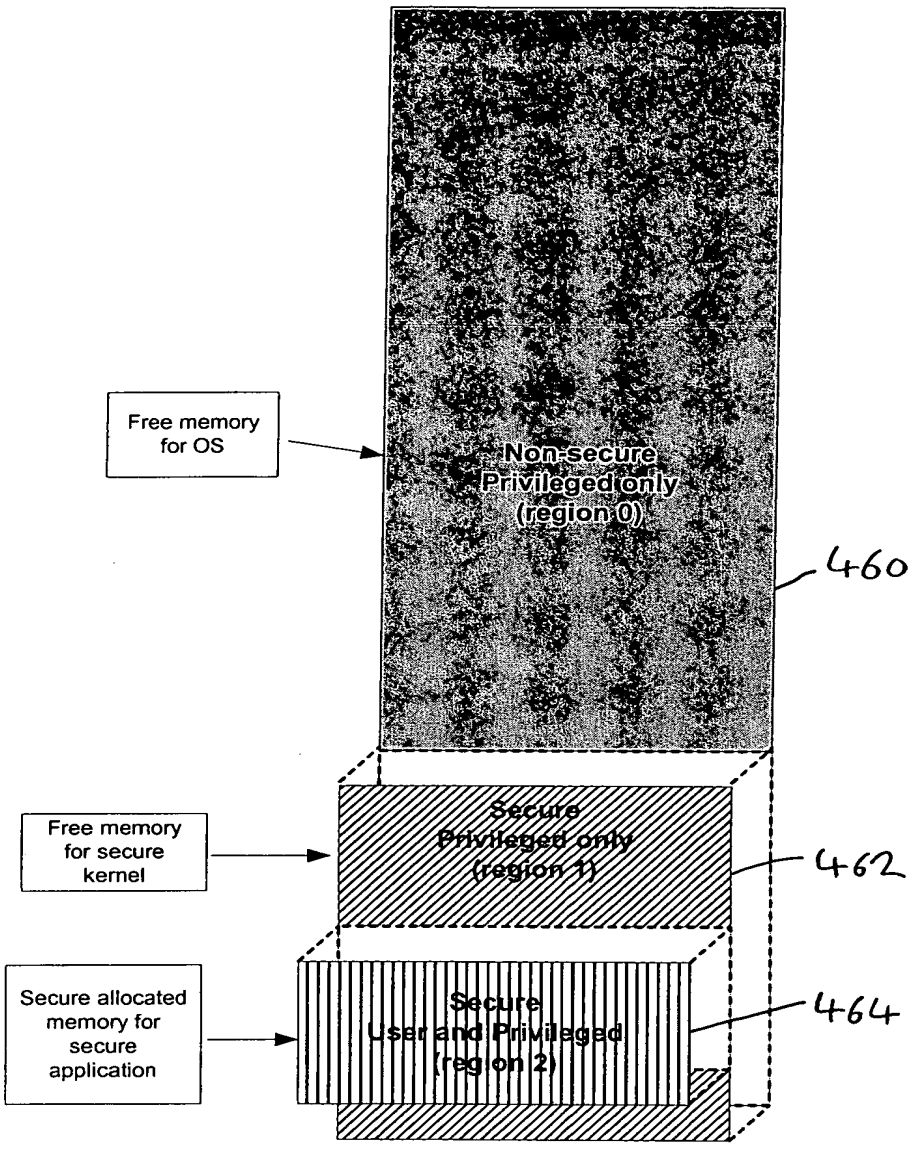

Free memory for OS
Non-secure Privileged only (region 0)
460
Free memory for secure kernel
Secure Privileged only (region 1)
462
Secure allocated memory for secure application
Secure User and Privileged (region 2)
464

FIG. 44

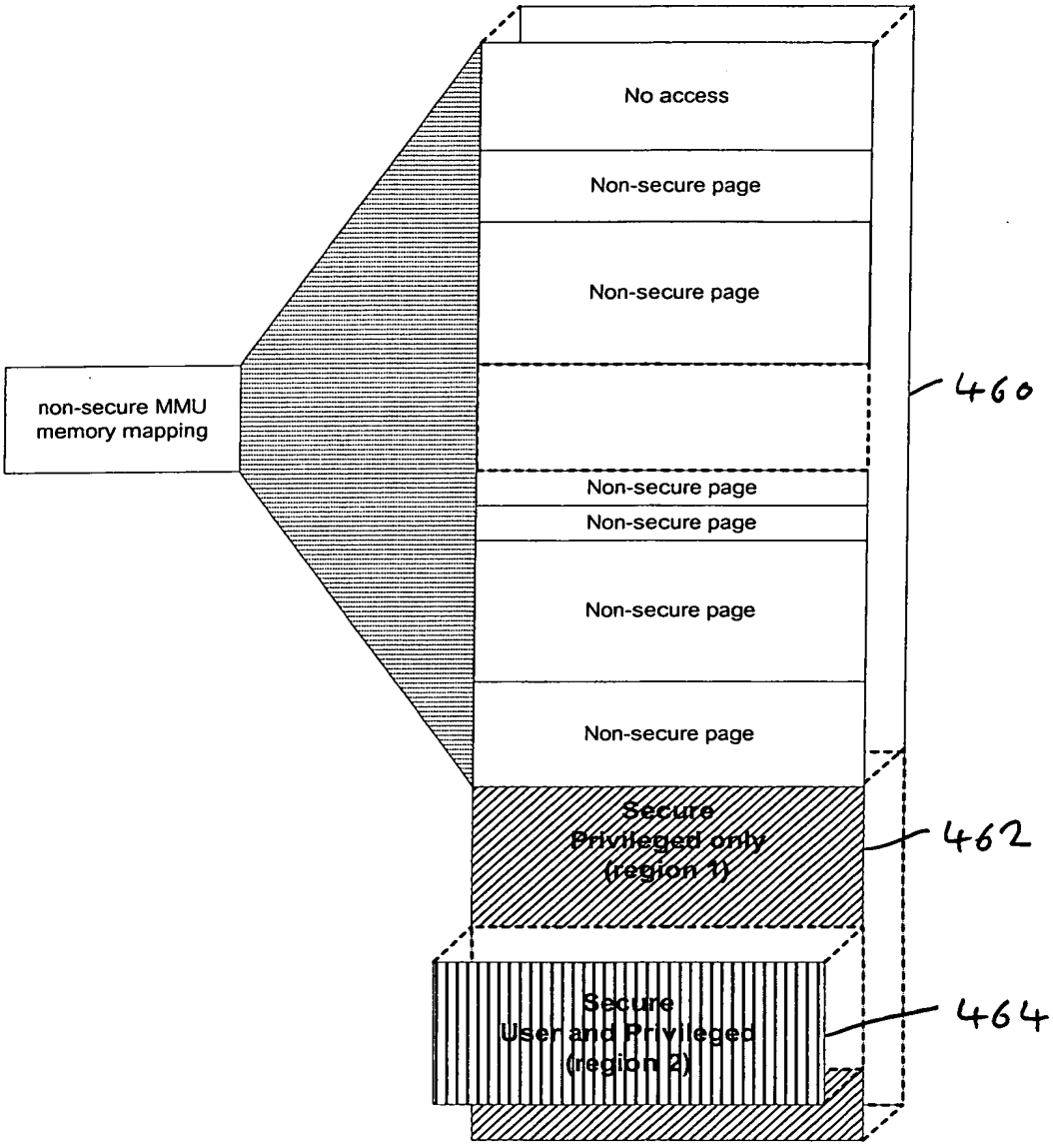

No access
Non-secure page
Non-secure page
non-secure MMU
memory mapping
Non-secure page
Non-secure page
Non-secure page
Non-secure page
Secure
Privileged only
(region 1)
Secure
User and Privileged
(region 2)
460
462
464

FIG. 45

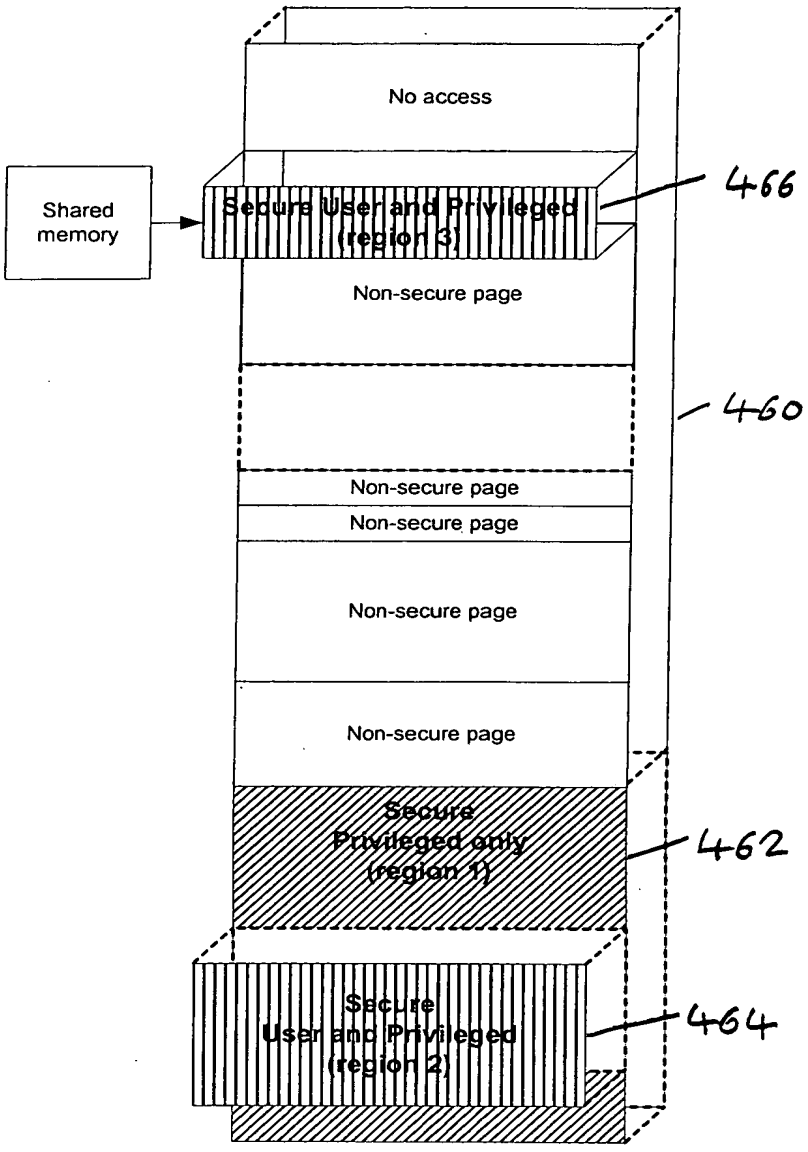

No access
Shared memory
Secure User and Privileged (region 3)
466
Non-secure page
460
Non-secure page
Non-secure page
Non-secure page
Non-secure page
Secure Privileged only (region 1)
462
Secure User and Privileged (region 2)
464

FIG. 4b

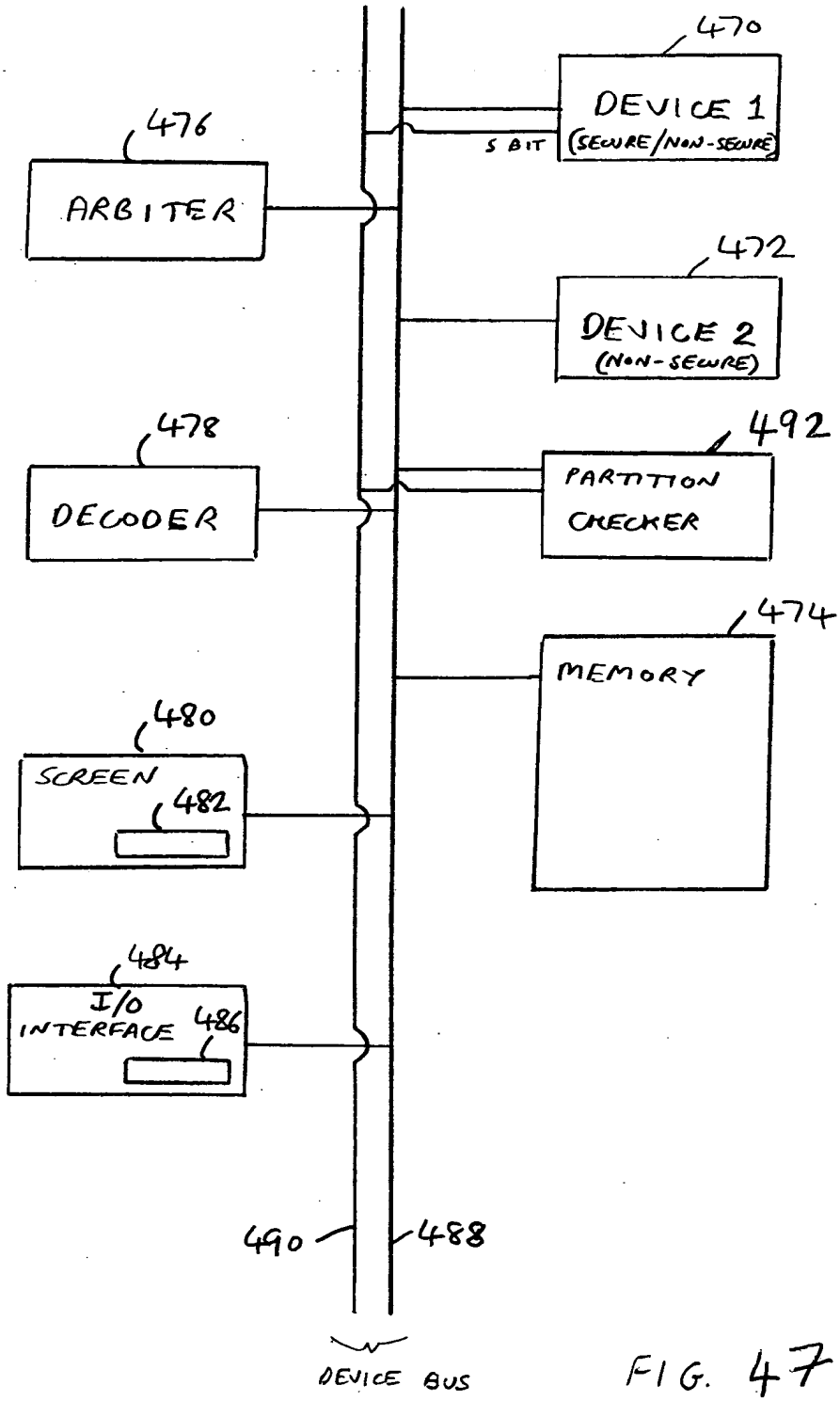

470
DEVICE 1
(SECURE/NON-SECURE)
5 BIT
476
ARBITER
472
DEVICE 2
(NON-SECURE)
478
DECODER
492
PARTITION
CHECKER
474
MEMORY
480
SCREEN
482
484
I/O
INTERFACE
486
490
488
DEVICE BUS

FIG. 47

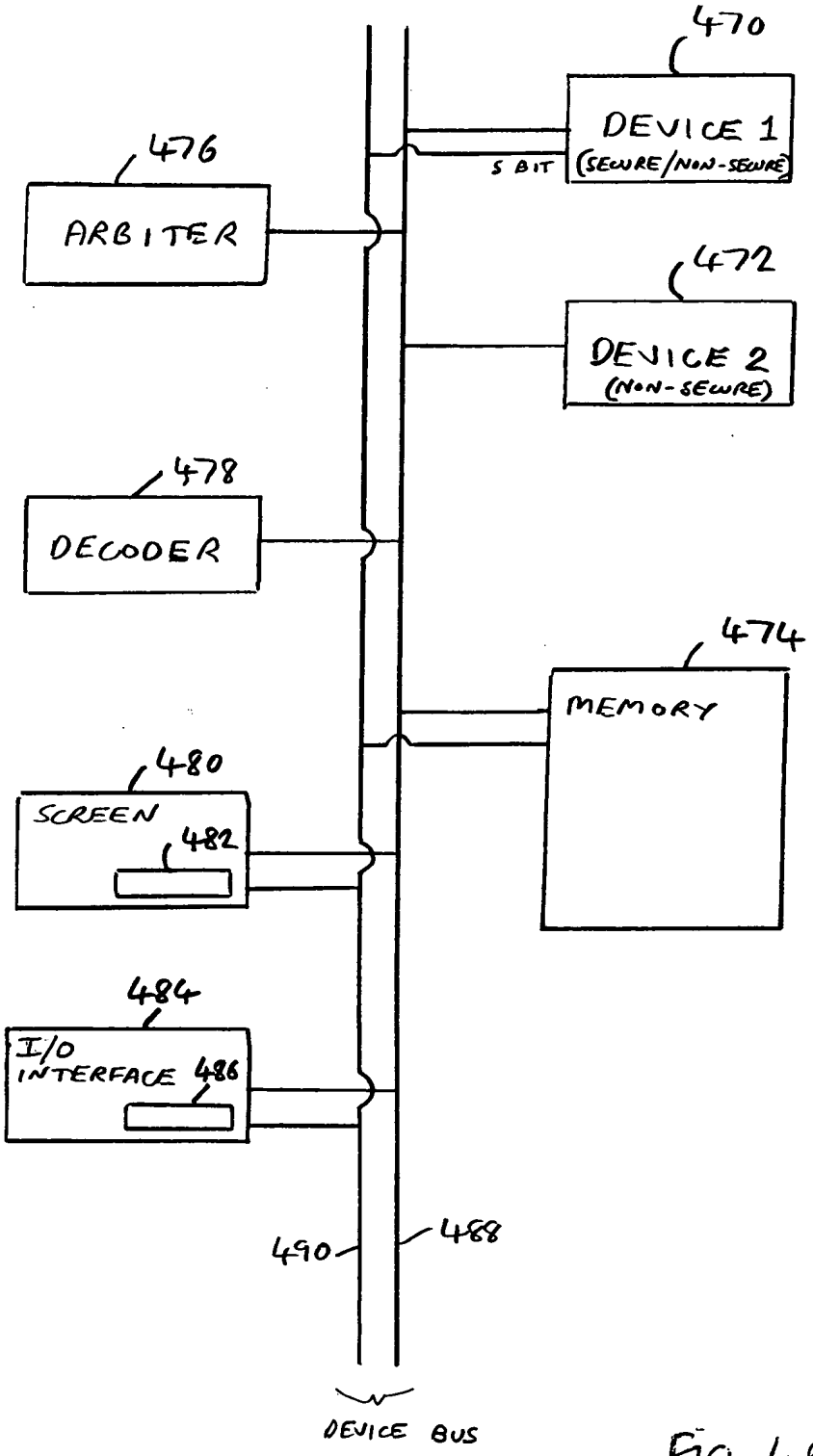

470
DEVICE 1
(SECURE/NON-SECURE)
5 BIT
476
ARBITER
472
DEVICE 2
(NON-SECURE)
478
DECODER
474
MEMORY
480
SCREEN
482
484
I/O
INTERFACE
486
490
488
DEVICE BUS

Fig. 48

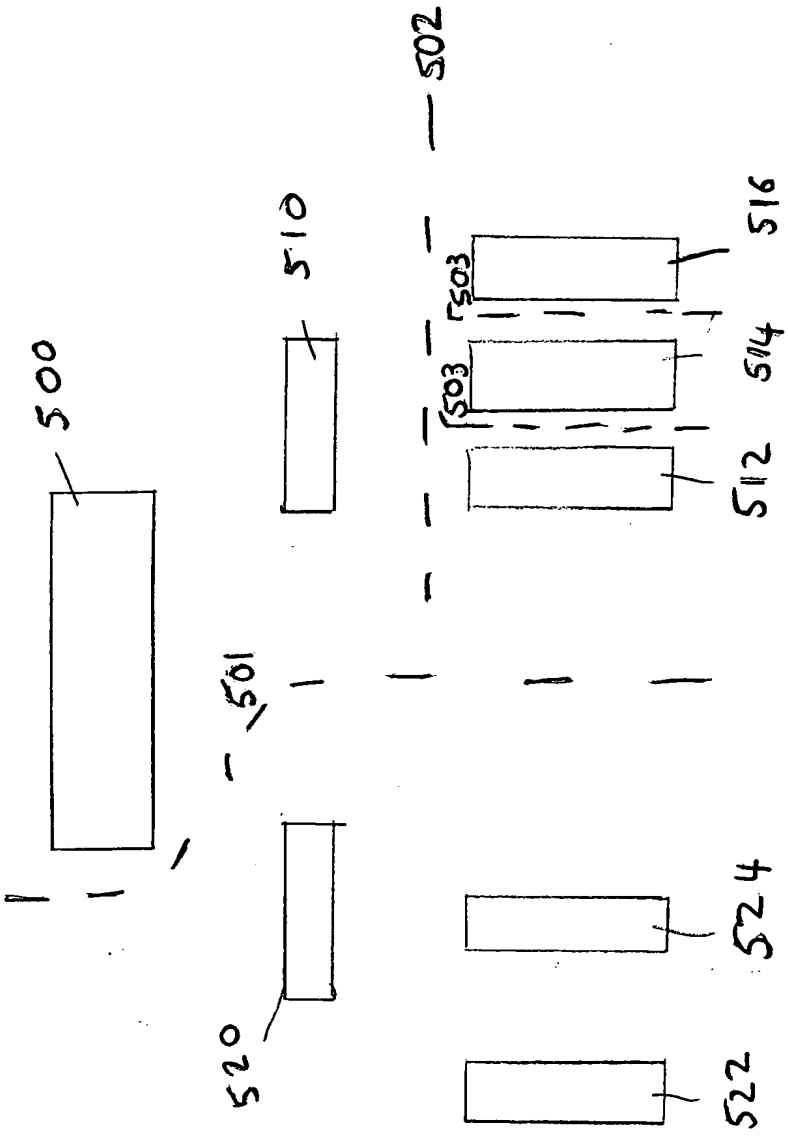


Figure 59

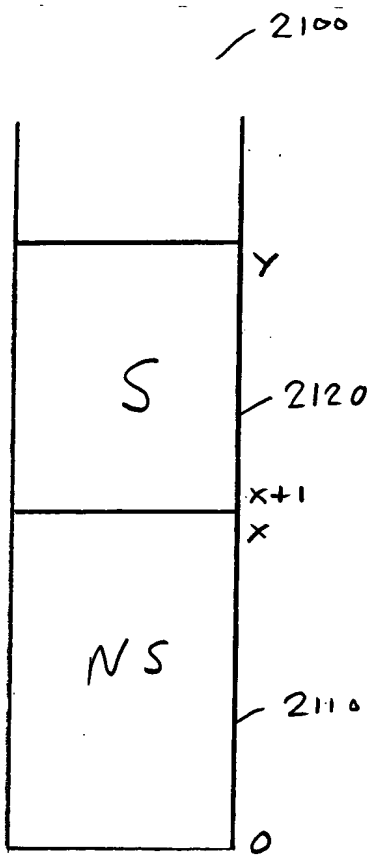


FIG. 49

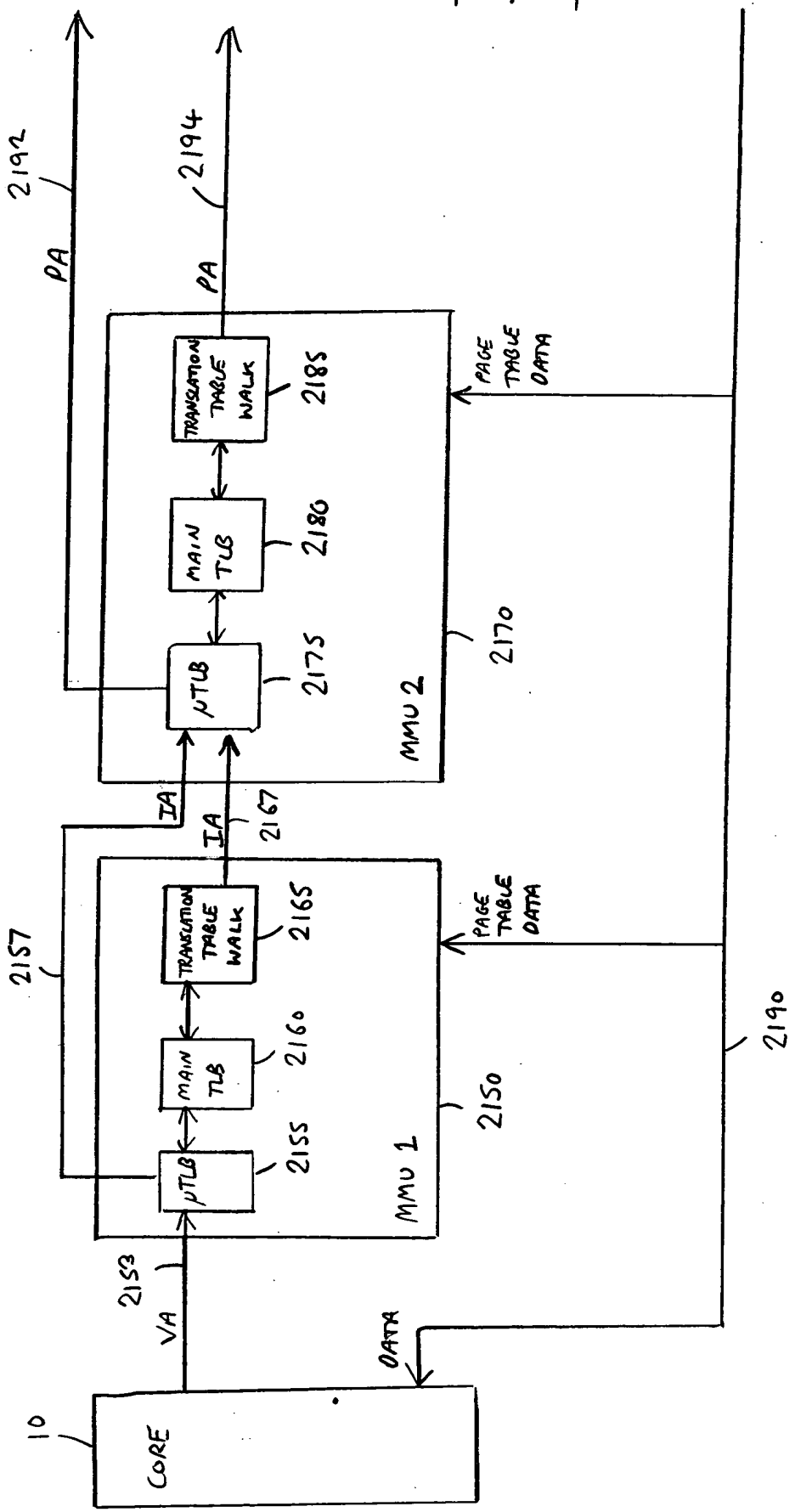

2192
PA
2194
PA
TRANSLATION TABLE WALK
2185
MAIN TLB
2180
μTLB
2175
MMU 2
2170
PAGE TABLE DATA
IA
IA
2167
2157
TRANSLATION TABLE WALK
2165
MAIN TLB
2160
μTLB
2155
MMU 1
2150
PAGE TABLE DATA
2190
2153
VA
DATA
10
CORE

FIG 50A

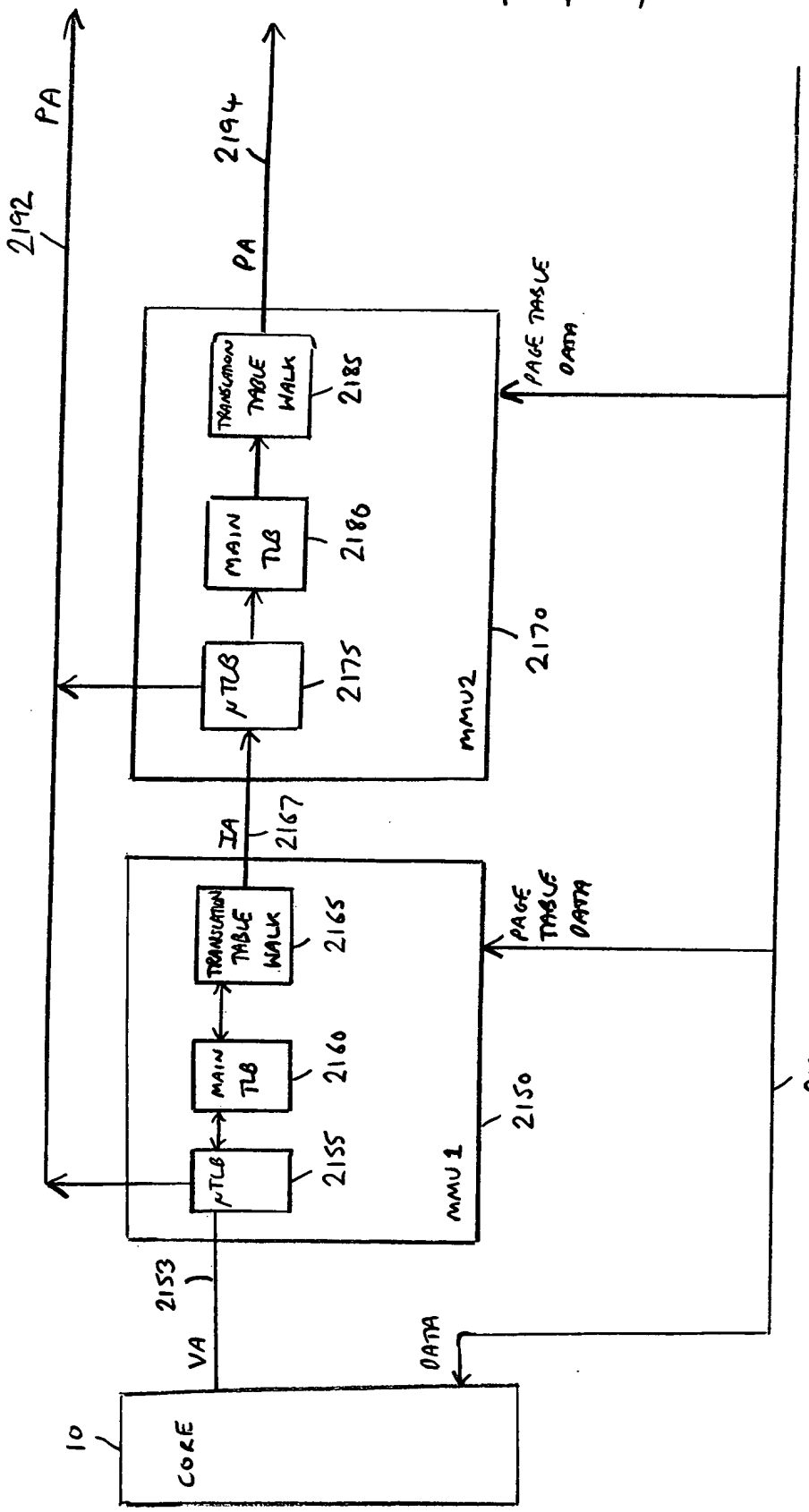

10
CORE
VA
2153
DATA
μTLB
2155
MAIN TLB
2160
TRANSLATION TABLE WALK
2165
MMU 1
2150
IA
2167
μTLB
2175
MAIN TLB
2186
TRANSLATION TABLE WALK
2185
MMU2
2170
PA
2192
PA
2194
PAGE TABLE DATA
PAGE TABLE DATA
2190

FIG 50B

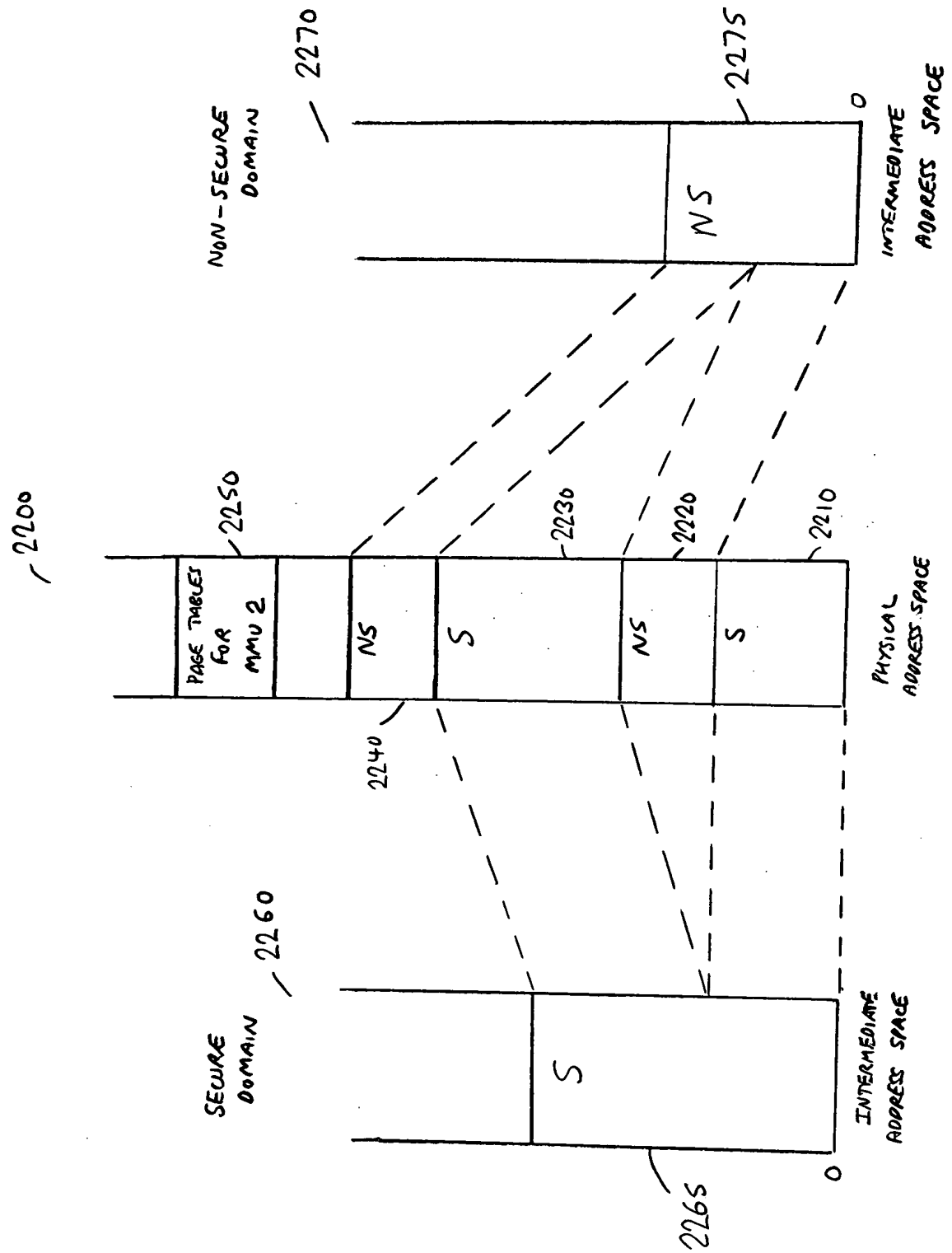

2200
2270
NON-SECURE DOMAIN
2275
0
NS
INTERMEDIATE ADDRESS SPACE
2250
PAGE TABLES FOR MMU 2
2230
2220
2210
NS
S
NS
S
PHYSICAL ADDRESS SPACE
2240
2260
SECURE DOMAIN
S
INTERMEDIATE ADDRESS SPACE
0
2265

FIG 51

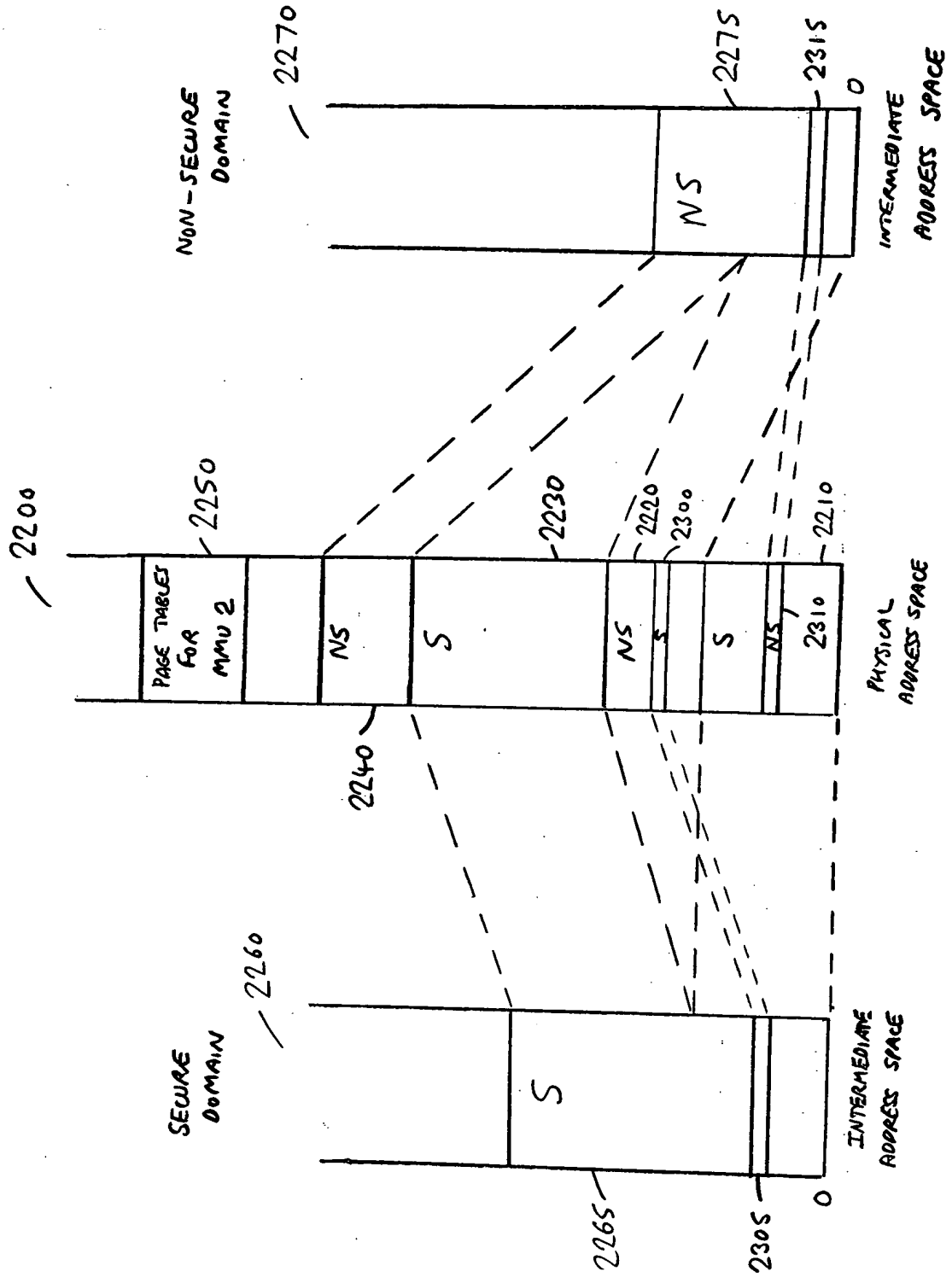

2200
2250
2230
2220
2300
2210
2270
2275
2315
0
NON-SECURE DOMAIN
INTERMEDIATE ADDRESS SPACE
PAGE TABLES FOR MMU 2
NS
S
NS
S
NS
S
NS
2310
PHYSICAL ADDRESS SPACE
2240
2260
SECURE DOMAIN
S
INTERMEDIATE ADDRESS SPACE
2265
2305
0

FIG 52

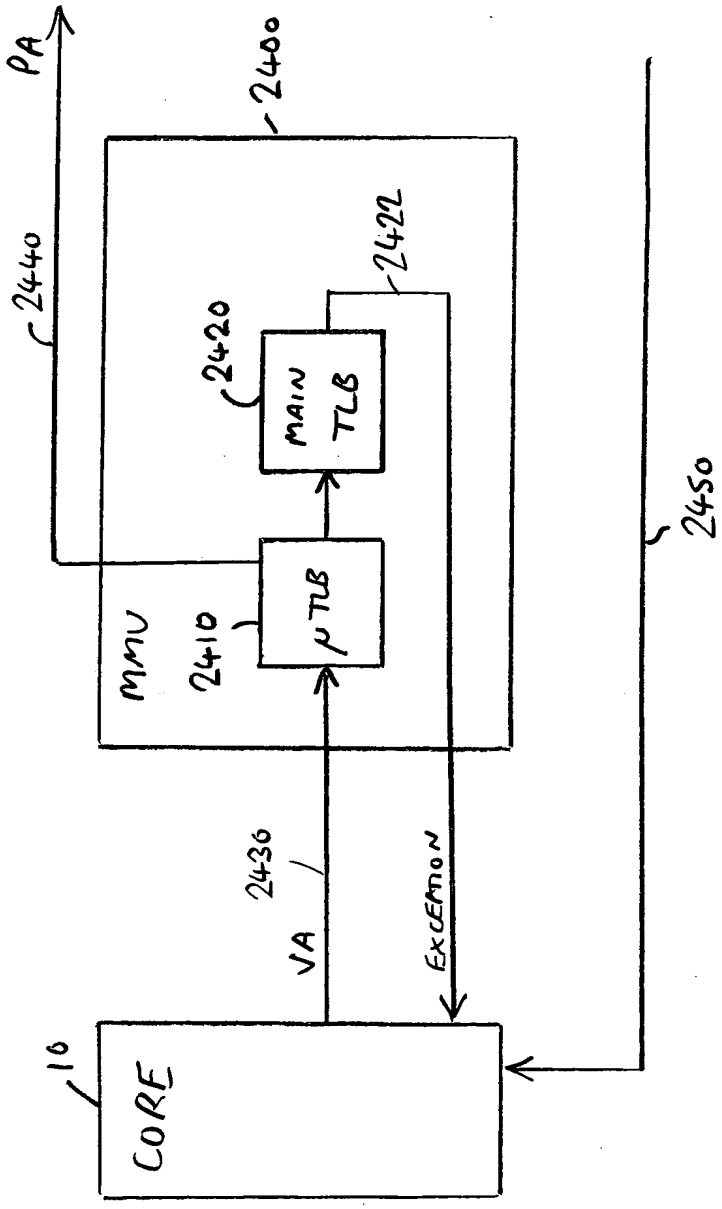

PA
2400
2440
2422
2420
MAIN TLB
MMU
2410
μTLB
2450
2436
VA
EXCEPTION
10
CORE

FIG 53

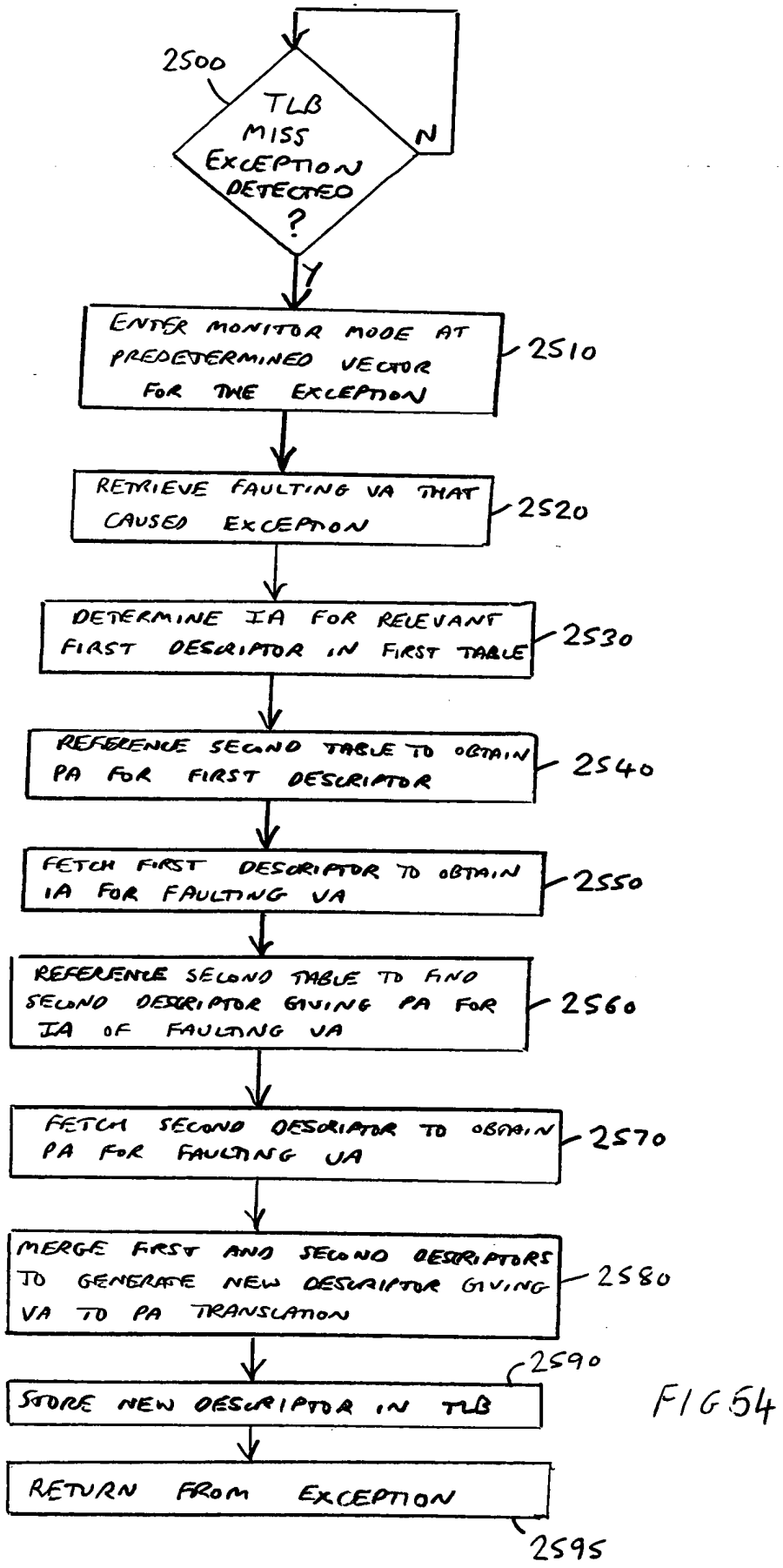

2500
TLB MISS EXCEPTION DETECTED ?
N
Y
ENTER MONITOR MODE AT PREDETERMINED VECTOR FOR THE EXCEPTION
2510
RETRIEVE FAULTING VA THAT CAUSED EXCEPTION
2520
DETERMINE IA FOR RELEVANT FIRST DESCRIPTOR IN FIRST TABLE
2530
REFERENCE SECOND TABLE TO OBTAIN PA FOR FIRST DESCRIPTOR
2540
FETCH FIRST DESCRIPTOR TO OBTAIN IA FOR FAULTING VA
2550
REFERENCE SECOND TABLE TO FIND SECOND DESCRIPTOR GIVING PA FOR IA OF FAULTING VA
2560
FETCH SECOND DESCRIPTOR TO OBTAIN PA FOR FAULTING VA
2570
MERGE FIRST AND SECOND DESCRIPTORS TO GENERATE NEW DESCRIPTOR GIVING VA TO PA TRANSLATION
2580
2590
STORE NEW DESCRIPTOR IN TLB
RETURN FROM EXCEPTION
2595

FIG 54

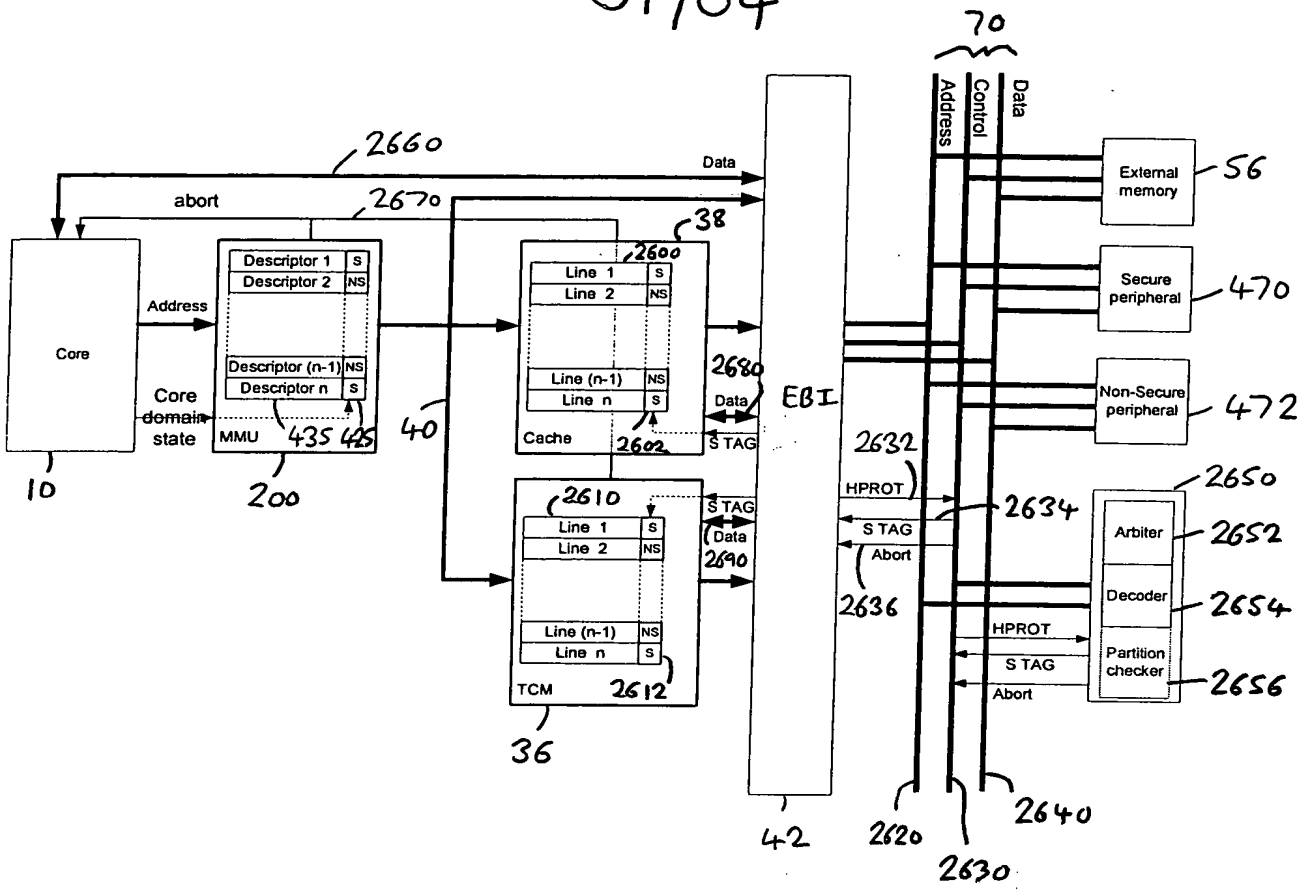

70
Address
Control
Data
2660
Data
abort
2670
38
External memory
56
Secure peripheral
470
Non-Secure peripheral
472
Core
Address
Descriptor 1
Descriptor 2
Descriptor (n-1)
Descriptor n
S
NS
Core domain state
MMU
435
425
200
10
40
2600
Line 1
Line 2
Line (n-1)
Line n
Cache
2602
2680
Data
S TAG
EBI
2610
TCM
2612
36
42
2632
HPROT
S TAG
Abort
2636
2634
2650
Arbiter
2652
Decoder
2654
Partition checker
2656
2620
2630
2640

FIG 55

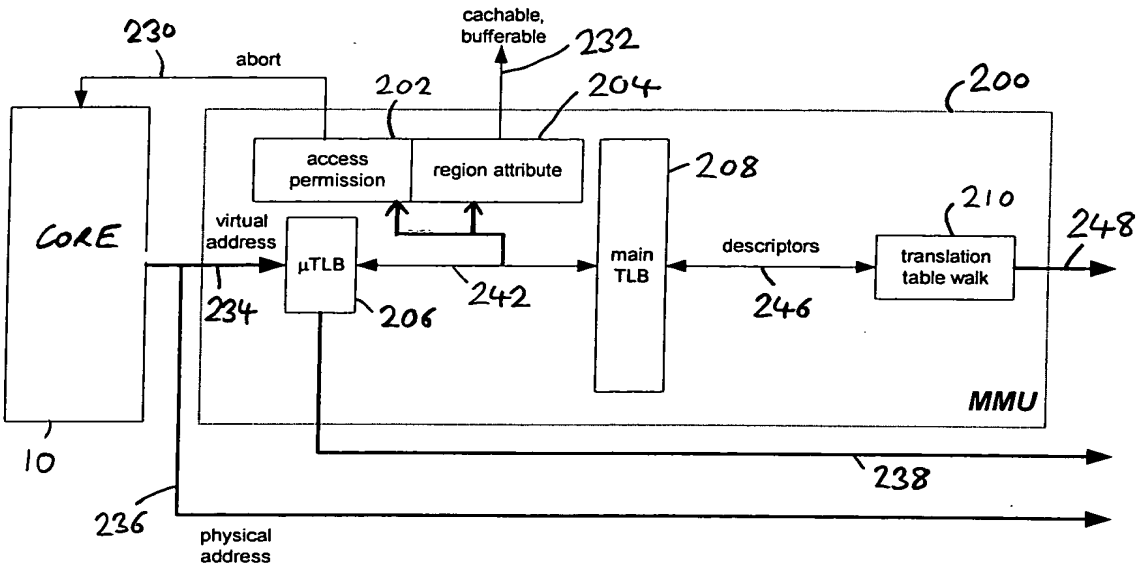

230
abort
cachable,
bufferable
232
202
204
200
access
permission
region attribute
208
210
248
CORE
virtual
address
µTLB
main
TLB
descriptors
translation
table walk
234
206
242
246
MMU
10
238
236
physical
address

FIG 5G

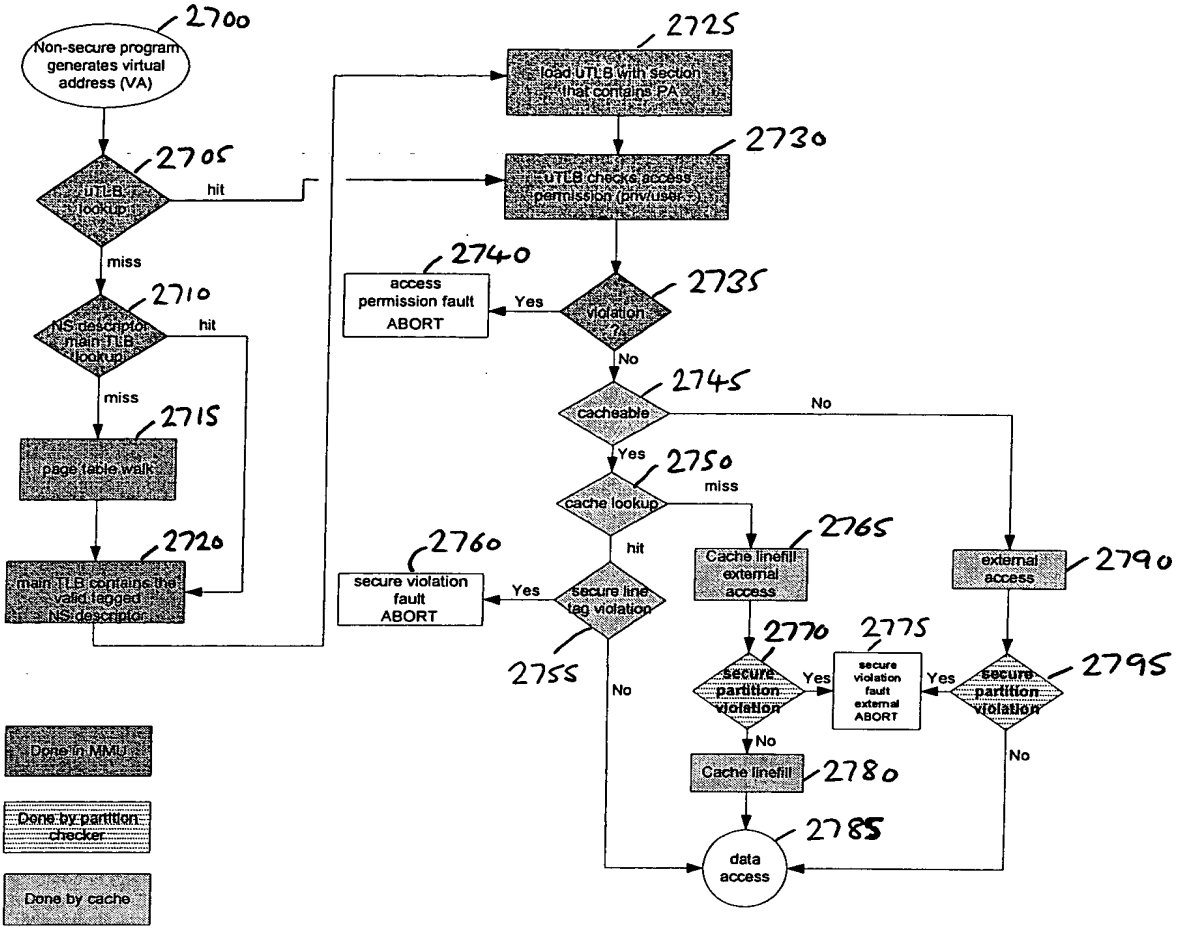

2700
Non-secure program generates virtual address (VA)
2705
UTLB lookup
hit
miss
2710
NS descriptor main TLB lookup
hit
miss
2715
page table walk
2720
main TLB contains the valid tagged NS descriptor
2725
load UTLB with section that contains PA
2730
UTLB checks access permission (priv/user)
2740
access permission fault ABORT
Yes
2735
violation
No
2745
cacheable
No
Yes
2750
cache lookup
miss
hit
2760
secure violation fault ABORT
Yes
secure line tag violation
2755
No
2765
Cache linefill external access
2770
secure partition violation
Yes
2775
secure violation fault external ABORT
Yes
2790
external access
2795
secure partition violation
No
No
Cache linefill
2780
2785
data access
Done in MMU
Done by partition checker
Done by cache

FIG 57

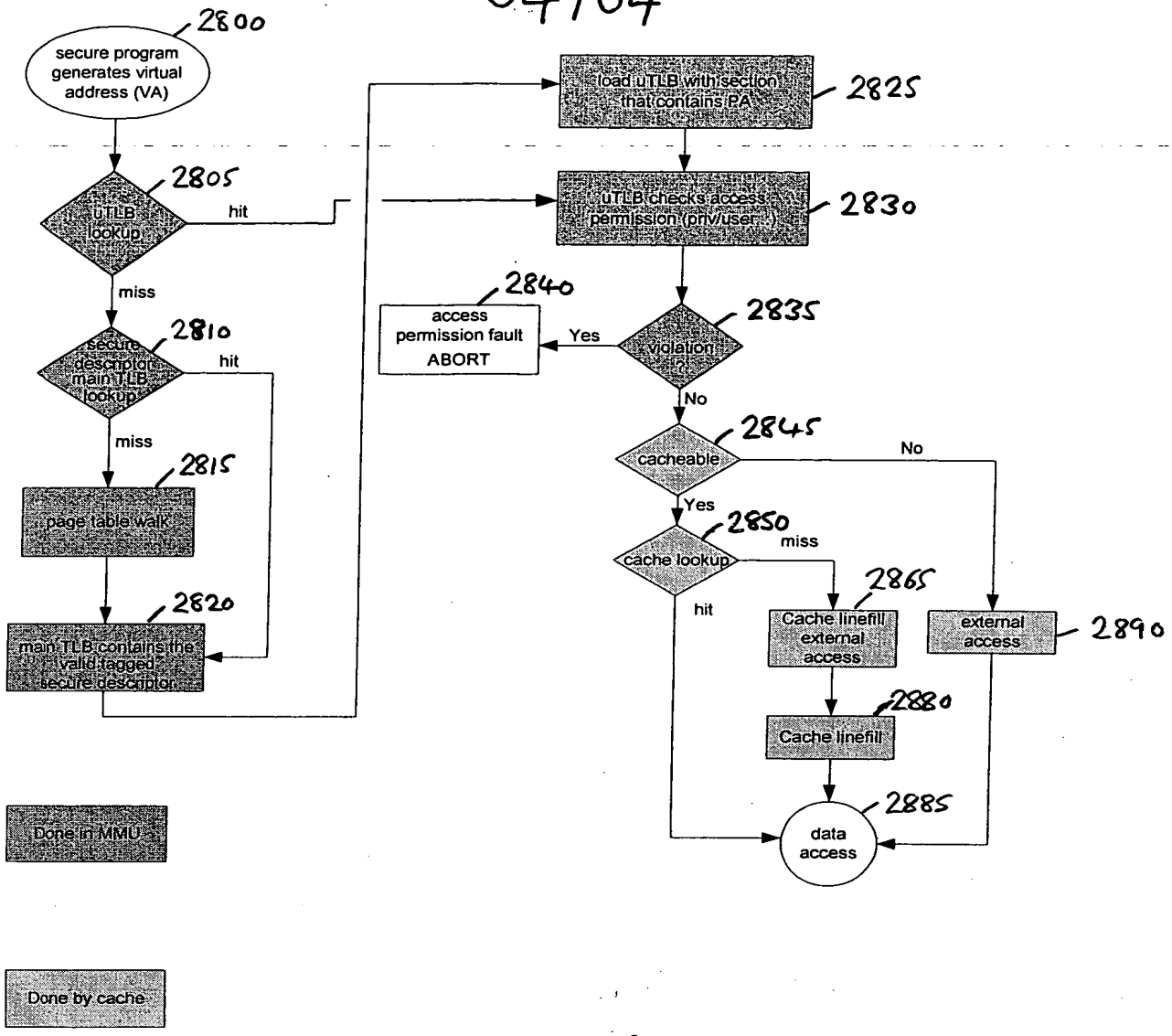

2800
secure program generates virtual address (VA)
2805
uTLB lookup
hit
miss
2810
secure descriptor main TLB lookup
hit
miss
2815
page table walk
2820
main TLB contains the valid tagged secure descriptor
2825
load uTLB with section that contains PA
2830
uTLB checks access permission (priv/user...)
2835
violation ?
Yes
2840
access permission fault ABORT
No
2845
cacheable
No
Yes
2850
cache lookup
miss
hit
2865
Cache linefill external access
2880
Cache linefill
external access
2890
2885
data access
Done in MMU
Done by cache

FIG 58

Method of entry	How to program?	How to enter?	Entry mode
Breakpoint hits	Debug TAP or software (CP14)	Program breakpoint register and/or context-ID register and comparisons succeed with Instruction Address and/or CP15 Context ID ([2]).	Halt/monitor ([1])
Software breakpoint instruction	Put a BKPT instruction into scan chain 4 (Instruction Transfer Register) through Debug TAP or Use BKPT instruction directly in the code.	BKPT instruction must reach execution stage.	Halt/monitor
Vector trap breakpoint	Debug TAP	Program vector trap register and address matches.	Halt/monitor
Watchpoint hits	Debug TAP or software (CP14)	Program watchpoint register and/or context-ID register and comparisons succeed with Instruction Address and/or CP15 Context ID ([2]).	Halt/monitor ([1])
Internal debug request	Debug TAP	Halt instruction has been scanned in.	Halt
External debug request	Not applicable	EDBGRQ input pin is asserted.	Halt

([1]): In monitor mode, breakpoints and watchpoints cannot be data-dependent.

([2]): The cores have support for thread-aware breakpoints and watchpoints in order to enable secure debug on some particular threads.

Figure 60

Name	Meaning	Reset value	Access	Inserted in scan chain for test
Monitor mode enable bit	0: halt mode 1: monitor mode	1	R/W by programming the ICE by the JTAG (scan1) • R/W by using MRC/MCR instruction (CP14)	yes
Secure debug enable bit	0: debug in non-secure world only. 1: debug in secure world and non-secure world	0	In functional mode or debug monitor mode: R/W by using MRC/MCR instruction (CP14) (only in secure supervisor mode) In Debug halt mode: No access – MCR/MRC instructions have any effect. (R/W by programming the ICE by the JTAG (scan1) if JSDAEN=1	no
Secure trace enable bit	0: ETM is enabled in non-secure world only. 1: ETM is enabled in secure world and non-secure world	0	In functional mode or debug monitor mode: R/W by using MRC/MCR instruction (CP14) (only in secure supervisor mode) In Debug halt mode: No access – MCR/MRC instructions have any effect. (R/W by programming the ICE by the JTAG (scan1) if JSDAEN=1	no
Secure user-mode enable bit	0: debug is not possible in secure user mode 1: debug is possible in secure user mode	1	In functional mode or debug monitor mode: R/W by using MRC/MCR instruction (CP14) (only in secure supervisor mode) In Debug halt mode: No access – MCR/MRC instructions have any effect. (R/W by programming the ICE by the JTAG (scan1) if JSDAEN=1	no
Secure thread-aware enable bit	0: debug is not possible for a particular thread 1: debug is possible for a particular thread	0	In functional mode or debug monitor mode: R/W by using MRC/MCR instruction (CP14) (only in secure supervisor mode) In Debug halt mode: No access – MCR/MRC instructions have any effect. (R/W by programming the ICE by the JTAG (scan1) if JSDAEN=1	no

Figure 61

Function Table

D	CK	Q[n+1]
0	↑	0
1	↑	1
X	↓	Q[n]

Logic Symbol

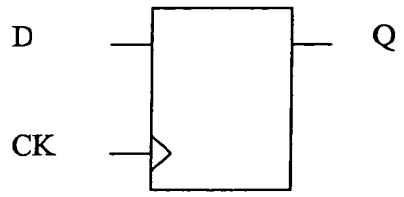


FIGURE 62

Function Table

D	SI	SE	CK	Q[n+1]
0	X	0	↑	0
1	X	0	↑	1
X	X	X	↓	Q[n]
X	0	1	↑	0
X	1	1	↑	1

Logic Symbol

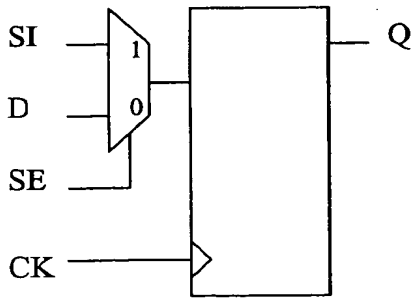


figure 63

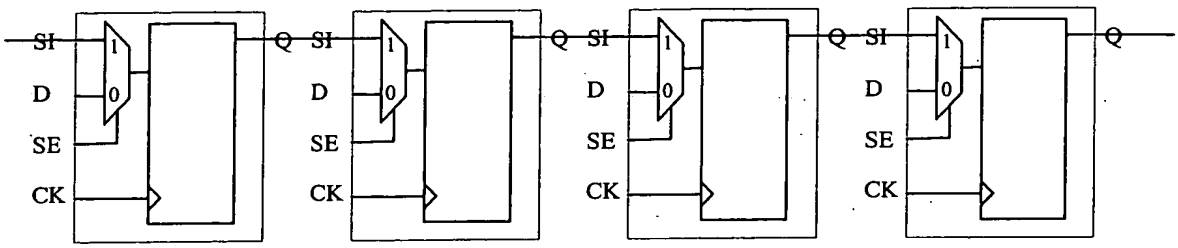


FIGURE 64

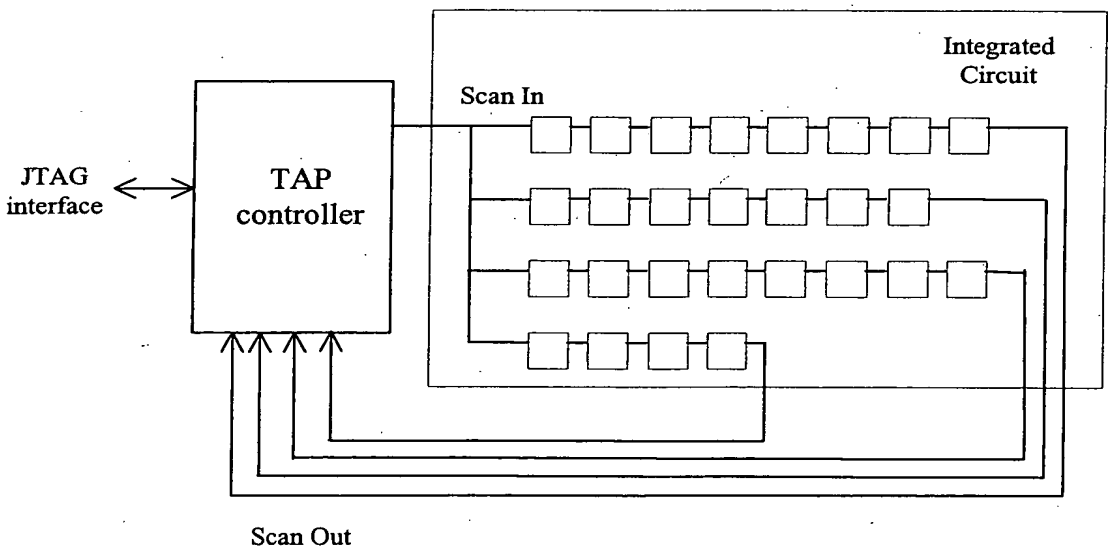


figure 65.

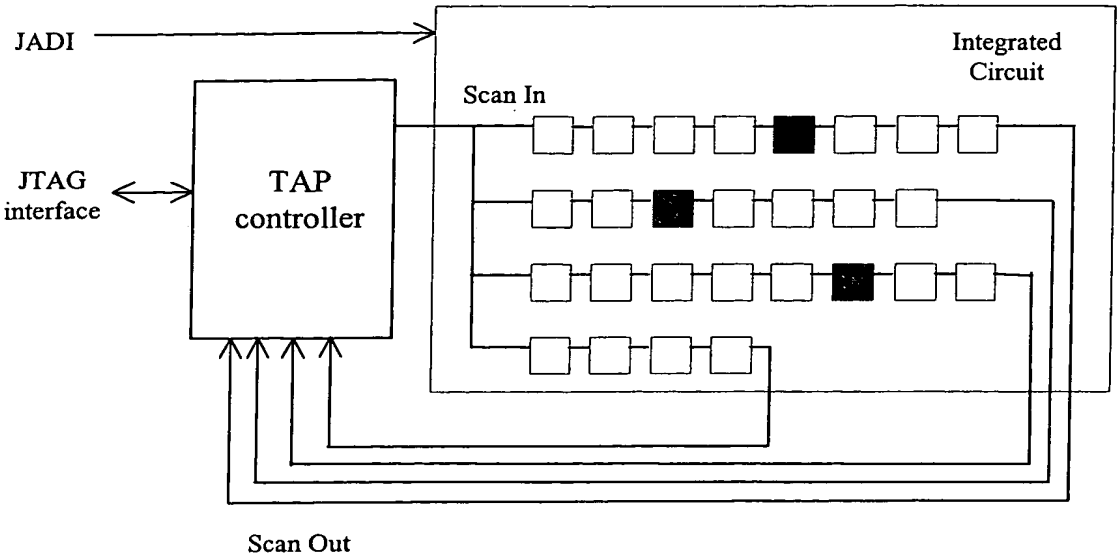


FIGURE 66A

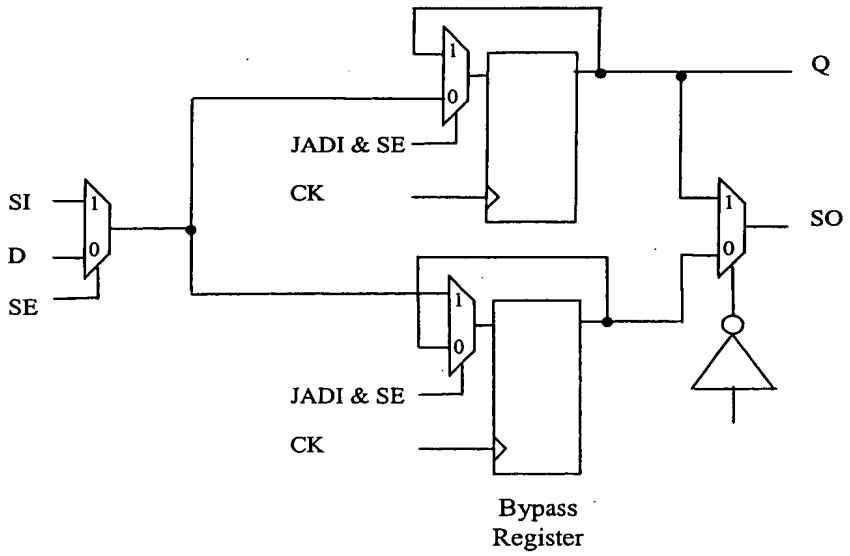


FIGURE 66B

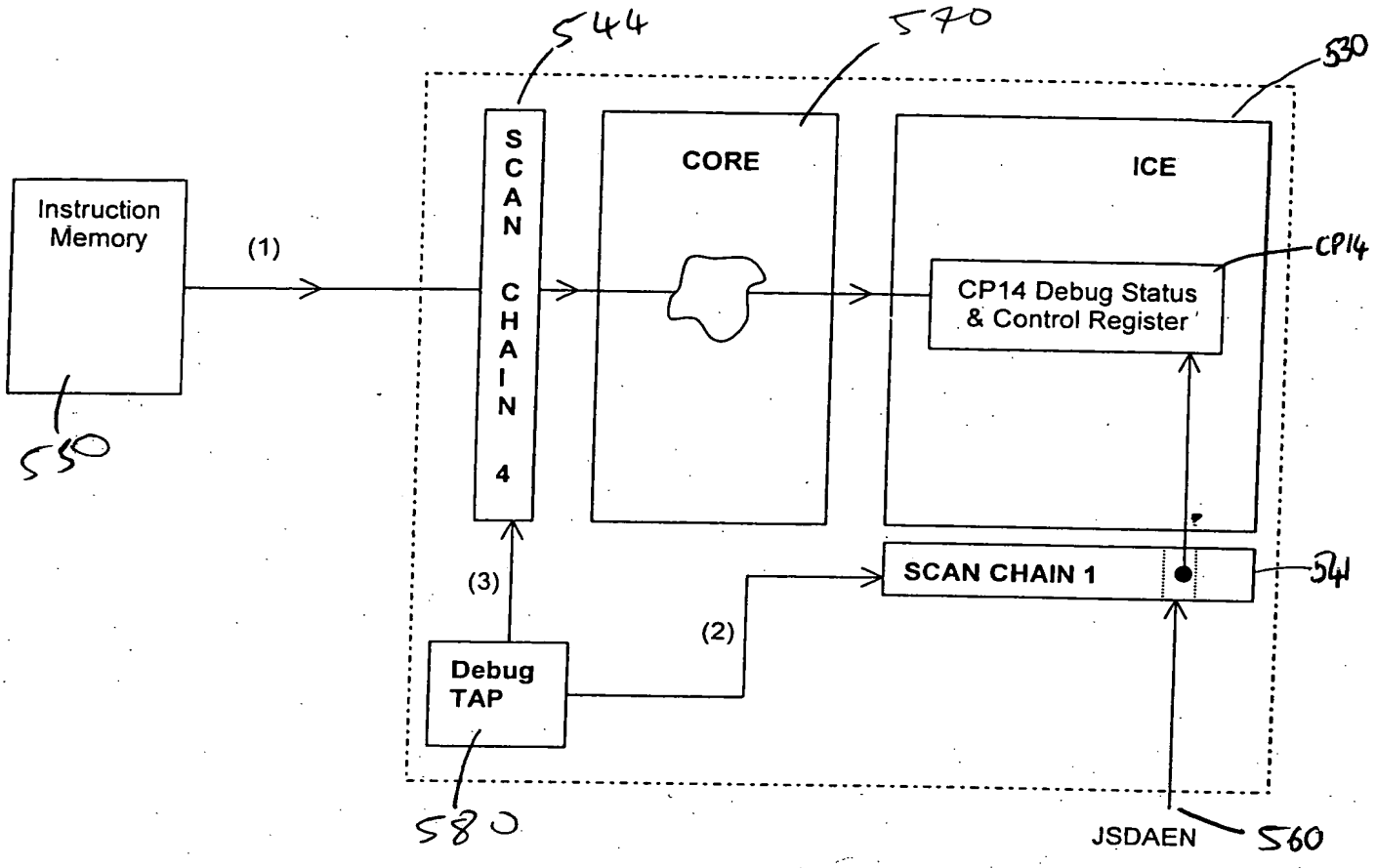

544
570
530
SCAN CHAIN 4
CORE
ICE
Instruction Memory
(1)
CP14
CP14 Debug Status & Control Register
550
(3)
SCAN CHAIN 1
541
(2)
Debug TAP
580
JSDAEN
560

Figure 67

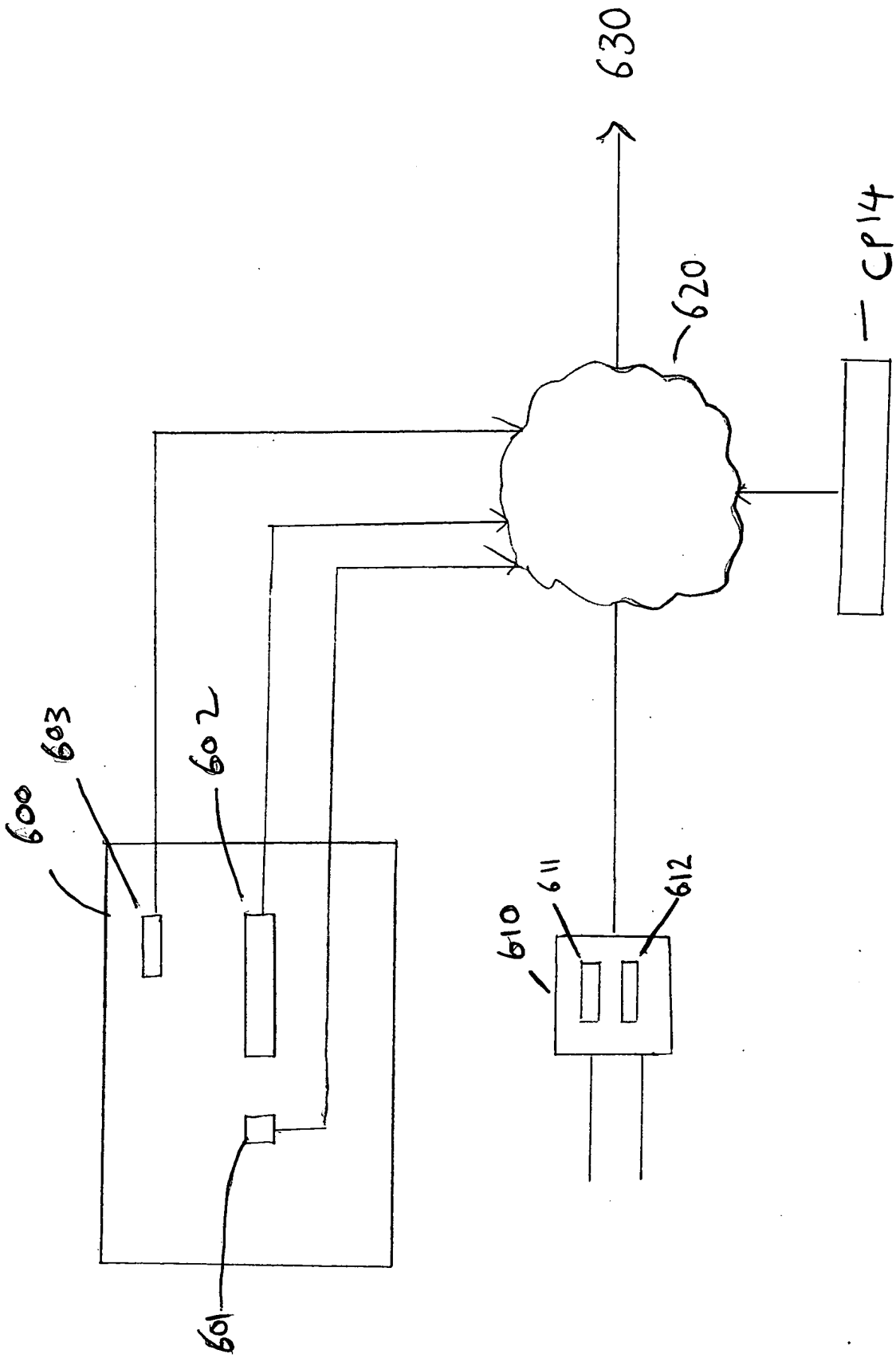


Figure 68

CP14 bits in Debug and Status Control register			meaning
Secure debug enable bit	Secure user-mode debug enable bit	Secure thread-aware debug enable bit	
0	X	X	No intrusive debug in entire secure world is possible. Any debug request, breakpoints, watchpoints, and other mechanism to enter debug state are ignored in entire secure world.
1	0	X	Debug in entire secure world is possible
1	1	0	Debug in secure user-mode only. Any debug request, breakpoints, watchpoints, and other mechanism to enter debug state are taken into account in user mode only. (Breakpoints and watchpoints linked or not to a thread ID are taken into account). Access in debug is restricted to what secure user can have access to.
1	1	1	Debug is possible only in some particular threads. In that case only thread-aware breakpoints and watchpoints linked to a thread ID are taken into account to enter debug state. Each thread can moreover debug its own code, and only its own code.

Figure 69A

CP14 bits in Debug and Status Control register			meaning
Secure trace enable bit	Secure user-mode debug enable bit	Secure thread-aware debug enable bit	
0	X	X	No observable debug in entire secure world is possible. Trace module (ETM) must not trace internal core activity.
1	0	X	Trace in entire secure world is possible
1	1	0	Trace is possible when the core is in secure user-mode only.
1	1	1	Trace is possible only when the core is executing some particular threads in secure user mode. Particular hardware must be dedicated for this, or re-use breakpoint register pair: Context ID match must enable trace instead of entering debug state.

Figure 69B

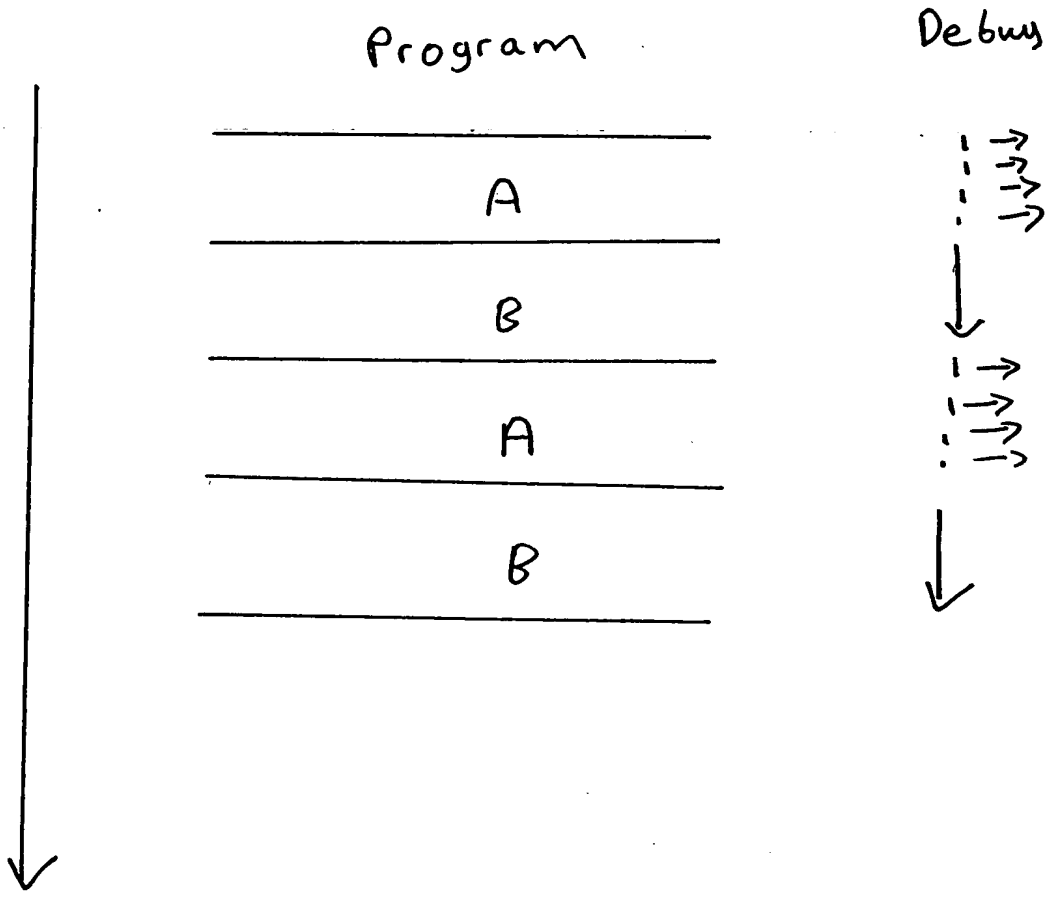


Figure 70

Method of entry	Entry when in non-secure world	entry when in secure world
Breakpoint hits	Non-secure prefetch abort handler	secure prefetch abort handler
Software breakpoint instruction	Non-secure prefetch abort handler	secure prefetch abort handler
Vector trap breakpoint	Disabled for non-secure data abort and non-secure prefetch abort interruptions. For other non-secure exceptions, prefetch abort.	Disabled for secure data abort and secure prefetch abort exceptions ([1]). For other exceptions, secure prefetch abort.
Watchpoint hits	Non-secure data abort handler	secure data abort handler
Internal debug request	Debug state in halt mode	debug state in halt mode
External debug request	Debug state in halt mode	debug state in halt mode

(i) see information on vector trap register,

(2) Note that when external or internal debug request is asserted, the core enters halt mode and not monitor mode.

Figure 71A

Method of entry	Entry in non-secure world	entry in secure world
Breakpoint hits	Non-secure prefetch abort handler	breakpoint ignored
Software breakpoint instruction	Non-secure prefetch abort handler	instruction ignored ([1])
Vector trap breakpoint	Disabled for non-secure data abort and non-secure prefetch abort interruptions. For others interruption non-secure prefetch abort.	breakpoint ignored
Watchpoint hits	Non-secure data abort handler	watchpoint ignored
Internal debug request	Debug state in halt mode	request ignored
External debug request	Debug state in halt mode	request ignored
Debug re-entry from system speed access	not applicable	not applicable

([1]) As substitution of BKPT instruction in secure world from non-secure world is not possible, non-secure abort must handle the violation.

Figure 71B